Acceleration Techniques for Photo Realistic Computer Generated Integral Images

Osama Hassan Youssef

Ph.D.

DE MONTFORT UNIVERSITY

April 2004

Acceleration Techniques for Photo Realistic Computer Generated Integral Images

Submitted in fulfillment of the requirements for the degree of Doctor of Philosophy

Osama Hassan Youssef

3D Imaging Group
Faculty of Computing Sciences and Engineering
DE MONTFORT UNIVERSITY, UK

April 2004

بسم الله الرحمن الرحيم[1]

To my family and my brothers

[1] An Arabic phrase which means: In the Name of Allâh, the Most Gracious, the Most Merciful

Acknowledgements

All the praises and thanks are to Allâh, the Lord of all that exists, the Most Gracious, the Most Merciful, the Owner of the Day of Recompense, who in his infinite mercy, blessed me with unlimited bounties. One of these bounties is his help to finish my research. I ask Allah to mention in the highest place and lay his peace upon the one who taught me the most valuable thing in life, Islam, and encouraged me to learn and to serve the humanity, the last prophet Muhammad.

I would like to take this opportunity to thank my supervisors Dr. Amar Aggoun and Prof. Malcolm McCormick for their support, help and criticism throughout the period of my research.

Many thanks are due to Prof. Wayne Wolf from Princeton university (USA), the collaboration with him and his feedback on this research are highly appreciated.

I also would like to extend my thanks to rest of the 3D Imaging group members, the staff of De Montfort university, and the staff of Princeton university for their help and support throughout the period of my studies.

Special thanks and appreciation go to Sheikh Abu Abdelrahman for his invaluable advice and wisdom .

I would like to thank John Stone, for his generosity of putting his ray tracing code

Available on the internet for researchers.

I gratefully acknowledge the support of my good friends here in UK and in Egypt.

I am thankful to my sisters and my brother for encouraging and supporting me to finish my postgraduate studies.

Last, but not least, I am deeply grateful to my mother, father, who have continuously supported and encouraged me not only throughout my education but throughout my life. They are giving me what I am not able to pay for. May Allâh reward them.

Abstract

The research work presented in this thesis has approached the task of accelerating the generation of photo-realistic integral images produced by integral ray tracing.

Ray tracing algorithm is a computationally exhaustive algorithm, which spawns one ray or more through each pixel of the pixels forming the image, into the space containing the scene. Ray tracing integral images consumes more processing time than normal images. The unique characteristics of the 3D integral camera model has been analysed and it has been shown that different coherency aspects than normal ray tracing can be investigated in order to accelerate the generation of photo-realistic integral images.

The image-space coherence has been analysed describing the relation between rays and projected shadows in the scene rendered. Shadow cache algorithm has been adapted in order to minimise shadow intersection tests in integral ray tracing. Shadow intersection tests make the majority of the intersection tests in ray tracing. Novel pixel-tracing styles are developed uniquely for integral ray tracing to improve the image-space coherence and the performance of the shadow cache algorithm. Acceleration of the photo-realistic integral images generation using the image-space coherence information between shadows and rays in integral ray tracing has been achieved with up to 41% of time saving. Also, it has been proven that applying the new styles of pixel-tracing does not affect of the scalability of integral ray tracing running over parallel computers.

The novel integral reprojection algorithm has been developed uniquely through geometrical analysis of the generation of integral image in order to use the tempo-spatial coherence information within the integral frames. A new derivation of integral projection matrix for projecting points through an axial model of a lenticular lens has been established. Rapid generation of 3D photo-realistic integral frames has been achieved with a speed four times faster than the normal generation.

Statement of Originality

The work contained within this thesis is purely that of the author unless otherwise stated.

Osama Hassan Youssef

Contents

Acknowledgement i

Abstract ii

Statement of Originality iii

Chapter 1 Introduction 1

1.1 The research area 1

1.2 Scope of the thesis and original contribution 3

1.3 Outline of the thesis 5

Chapter 2 Overview of 3D Imaging Techniques 8

2.1 Three dimensional imaging systems 9

2.1.1 3D with glasses (stereoscopy) 9

2.1.2 Autostereoscopic displays 11

2.1.2.1 Holography 11

2.1.2.2 Volumetric displays 13

2.1.2.3 Two-view Display, Head-tracking Image System and Multi-view Display System 15

2.1.2.4 Integral imaging 17

2.2 Advanced integral imaging system 19

2.3 Summary 22

Chapter 3 Overview of Ray Tracing 25

3.1 Formulation of the Global Illumination Problem 26
3.2 Ray tracing 30
3.2.1 The virtual camera 32
3.2.2 Visible surface determination 33
3.2.3 Surface shading 33
3.2.3.1 Ambient light 34
3.2.3.2 Shadows 35
3.2.3.3 Diffuse shading 36
3.2.3.4 Specular highlights 37
3.2.3.5 Specular reflection 40
3.2.3.6 Refraction 40
3.2.4 Aliasing 42
3.2.5 Numerical precision and recursion 44
3.2.6 Surface texturing 44
3.2.6.1 Coordinate transformation 45
3.2.6.2 Texture lookup 45
3.2.6.3 Improving mapping quality 47
3.2.6.4 Image mapped textures 47
3.2.6.5 Procedural texturing 49
3.3 Computer generation of integral images using ray tracing 50
3.4 Summary 51

Chapter 4 Traditional Acceleration Techniques for Ray Tracing 54

4.1 A board classification 55
4.2 Faster intersections 56
4.2.1 Efficient intersection tests 56

4.2.2 Bounding volumes 56
4.2.3 Bounding volumes hierarchy 57
4.2.4 Space subdivision and spatial coherence 59
4.3 Fewer rays 61
4.4 Generalised rays 62
4.4.1 Cone tracing 62
4.4.2 Beam tracing 63
4.4.3 Pencil tracing 65
4.5 Summary 66

Chapter 5 Novel Pixel-tracing styles for Faster Shadow Cache in Integral Ray Tracing 68

5.1 Shadow caching 69
5.2 Cache calculations 70
5.3 Integral camera model implications 72
5.4 Analysis of the number of continuous cache hits for lenticular sheets 73
5.4.1 Style 1: Tracing of consecutive pixels in the horizontal direction ... 74
5.4.2 Style 2: Tracing similar pixels in adjacent lenses 79
5.4.3 Style 3:Tracing pixels in the vertical direction 81
5.5 Analysis of the number of continuous cache hits for micro-lens arrays 82
5.5.1 Style 1: Tracing of consecutive pixels 82
5.5.2 Style 2: Tracing similar pixels in adjacent lenses 83
5.6 Optimum pixel tracing for maximum number of continuous cache hits 83
5.6.1 Missing a cache hit while tracing pixels 83
5.6.2 Tracing pixels in two directions for lenticular sheet 85
5.6.3 Tracing pixels in two directions for micro-lens array 86

5.7 Tests and Results 87
5.7.1 Lenticular sheet 92
5.7.1 Micro-lens array 97
5.8 Shadow cache and parallel integral ray tracing 102
5.8.1 Shadow cache and parallelism 102
5.8.1.1 Multithreaded ray tracing 103
5.9 Summary 105

Chapter 6 Rapid Integral Ray Tracing Using Reprojection 108

6.1 Projection in integral ray tracing 109
6.1.1 Camera matrix 110
6.1.2 Integral projection matrix 112
6.1.3 Implementation of integral reprojection 113
6.1.4 The novel integral reprojection algorithm 119
6.2 Z-buffer and correct occlusion 121
6.3 Interpolation 123
6.4 Filling holes 124
6.5 Tests and results 127
6.6 Summary 135

Chapter 7 Conclusions and Further Work 137

7.1 Conclusions 137
7.2 Further work 139

References 141

Appendix A Mathematical Framework for Global Illumination 149

List of Publications 163

Chapter 1

Introduction

1.1 The research area

The development of three-dimensional (3D) imaging systems is a constant pursuit of the scientific community and entertainment industry [Moto95, Taub02, 3dcg04]. There is growing evidence that 3D computer displays will have the potential to establish a future mass-market in the fields of medical imaging, defense and avionics. One of the much discussed applications that exist for 3D computer display and imaging systems is virtual reality.

Many different approaches have been adopted in attempts to realize free viewing 3D systems [Okos76, McAl93]. Holographic systems can produce still images with high quality and give an extremely realistic reproduction of the spatial images but have difficulty in producing and displaying moving spatial images due to the requirements for the coherent light sources [Gabo48, Leit63, Outw99]. Recently, several groups have demonstrated autostereoscopic computer imaging systems [Actualdepth, DTI, DDD, 4D-Vision, Philips, Genex, Stereographics]. Most of them work on the principle of presenting multiple images to the viewer by use of temporal or spatial multiplexing of several discrete viewpoints to the eyes. The viewing effect depends on the viewpoint number, a higher quality experience being enjoyed when the number of viewpoints is

high. This creates the problem of simultaneously generating or capturing enough views in real time at an affordable cost which cause major difficulty in the displaying system. To date, integral imaging is a technique that is capable of creating and encoding a true volume spatial optical model of the object scene in the form of a planar intensity distribution by using unique optical components [McCo95, Okan98, Min01, Naeu01, Yano02].

An innovative and unique optical system for transferring full parallax three-dimensional images has been developed by the 3D Imaging Group at De Montfort University (DMU) [Davi94, McCo95]. The two-tier optical camera arrangement overcomes the image degradation caused by the two-stage recording process and allows direct spatially correct 3D image capture for orthoscopic display. Subsequently, Okano *et al.* reported an integral optical transmission array that produced 3D image data suitable for imaging by four multiplexed CCD elements, an arrangement previously proposed to achieve the required resolution by McCormick *et al.* [Arai98, McCo92]. To date researchers have concentrated on establishing interactive generation of photo-realistic images for 3D computer displays viewing. Photo-realistic images are generated by time exhaustive algorithms such as ray tracing and radiosity. In respect of stereoscopic systems a number of research groups have tackled interactive computer generation of stereo pairs [Wan04, Kita01]. However there are many data processing issues that require specialist solutions unique to integral imaging. One of these issues is the implication of integral camera model on the way a required scene is rendered, which is particularly useful to enable fast rendering of photo-realistic integral images. The work in this thesis addresses the particular issue of accelerating the computer generation of photo-realistic integral images produced by ray tracing algorithm.

1.2 Scope of the thesis and original contribution

The generation of photo-realistic integral images using ray tracing algorithm was first introduced by Cartwright [Cart00]. Ray tracing algorithm spawns one ray or more through each pixel of the pixels forming the image, into the space containing the scene. For each ray, intersections with objects of the scene are tested. These tests are termed as intersection tests. This operation is further extended at intersection points lay on the objects of the scene generating shadow, reflection, and transparency rays. Intersection tests are computationally and time exhaustive and overshadow everything else, accounting for the vast bulk of time consumed by ray tracing. In order to produce integral images with acceptable quality, they have to be generated with higher resolution than normal computer images. Consequently, ray tracing integral images consume more processing time than normal images.

To find out the way of accelerating integral ray tracing, two aspects are investigated: the reduction of the number of intersection tests using the image-space coherence information in order to accelerate the generation of one integral image and the reduction of the number of rays to be traced using the tempo-spatial coherence information in order to generate a sequence of integral frames.

In the first aspect an algorithm defined as shadow cache algorithm developed by Hains [Hain86] is adapted according to the unique geometrical properties associated with the computer generation of the integral image. The shadow cache algorithm reduces the number of intersection tests associated with shadow rays, which make the majority of the rays in ray tracing. An analysis of the relationship between rays and shadows is carried out in order to find out the optimal pixel-tracing style for integral ray tracing that leads to the fastest shadow intersection tests using shadow cache. Novel pixel-tracing styles are developed uniquely for integral ray tracing to improve the image-space coherence and the performance of the shadow cache algorithm. Experiments show that the proposed pixel-tracing styles lead to time improvement in integral ray tracing that

saves up to 41% of the processing time. The novel pixel-tracing styles are explained in the thesis as well as a study of the scalability of ray tracing algorithm on parallel computers and how the proposed pixel-tracing styles do not affect this scalability.

Following the coherent tracing of pixels, a novel approach termed as integral reprojection is developed uniquely through geometrical analysis of the generation of integral images in order to use the tempo-spatial coherence information within the integral frames. A new integral projection matrix is derived to project intersection points of the scene onto the image plane through the model describing a lenticular lens. The pixels of a new frame are generated using the information left over from the previous frame. Experiments show that the integral projection algorithm accelerates the generation of integral frames four times more than its normal execution speed. How the algorithm works is explained in the thesis. A number of strategies, which are adopted in order to correct artefacts caused by the reprojection approach, are also described.

The thesis contains the following original contributions:

1. A new way of analysing image-space coherence in 3D integral ray tracing through the relationship between spawned rays and projected shadows.
2. Novel pixel-tracing styles, which improve the ray-object coherence in integral ray tracing and the performance of shadow cache algorithm.
3. Achievement of accelerating the generation of photo-realistic integral images using the image-space coherence information.
4. A novel way of fast generation of 3D photo-realistic integral frames using reprojection of previous frames.
5. A new derivation of integral projection matrix for projecting points through an axial model of a lenticular lens.
6. Achievement of high speed of generating 3D photo-realistic integral frames.

1.3 Outline of the thesis

Chapter 1 introduces the subject of the research work, providing an overview of the computer generation of photo-realistic 3D integral images, the need for accelerating the integral ray tracing algorithm, and the scope and original contribution of this work.

Chapter 2 contains a brief historical overview of two 3D imaging techniques: stereoscopic and autostereoscopic display. It concentrates on the free view (autostereoscopic) system and provides a detailed description of the 3D integral imaging system developed by the 3D Imaging Group at De Montfort University.

Chapter 3 describes different aspects of the ray tracing algorithm used as a renderer in the proposed project as well as the integral camera model integrated in the integral ray tracing algorithm in order to generate photo-realistic integral images.

Chapter 4 presents traditional acceleration techniques for ray tracing stating how time-efficiency has been the focus of the research of ray tracing due to the exhaustive computational property of ray tracing. Integral ray tracing is an adapted version of ray tracing algorithm. Therefore traditional ray tracing acceleration techniques can be used in integral ray tracing.

In Chapter 5, novel pixel-tracing styles are developed in order to utilise shadow cache algorithm for integral ray tracing acceleration. Shadow cache works by keeping a cache of the most recent shadow-casting object for each light source. Image-space coherence is exploited in order to decrease the number of shadow intersection tests. Cache calculations are illustrated and showed that tracing pixels in a non-coherent way is very computationally expensive and could duplicate the computation cost of shadow rays in shadow cache algorithm. The structure of the lenses and the camera model in the integral ray tracing affects the way primary rays are spawned as well as the spatial

coherence among them. In order to find out the optimal pixel-tracing style, the relationship between pixels and their primary rays and shadows is analysed. The analysis consists of the way primary rays relate to the shadow size and location and its effect on the number of shadow intersection tests in integral ray tracing. Three main styles of tracing pixels for lenticular sheets are exploited: Style 1 Tracing of consecutive pixels in the horizontal direction, style 2 Tracing similar pixels in adjacent lenses, and style 3 Tracing pixels in the vertical direction. For micro-lens arrays two main styles are exploited: Tracing of consecutive pixels, and tracing similar pixels in adjacent lenses. Optimum pixel-tracing styles are achieved by merging different styles into new ones that work on pixels in two dimensions, both vertical and horizontal. Two-dimensional style for lenticular sheet traces strips of pixels that cover the maximum area of the projected shadow. For micro-lens array, square areas of pixels are traced in order to cover the maximum area of the projected shadow and accommodate the ray coherence. Experiments show that the proposed coherent pixel-tracing with respect to the unique properties of the modelled integral camera resulted in time-improving integral ray tracing by up to 41% for lenticular sheets and 18.6% for micro-lens arrays. Running integral ray tracing on parallel computers need some careful manipulations of the cached object that are explained in the chapter in order to avoid using mutual exclusion locks that severely degrade the time-performance. This helps the execution on multiprocessor computers and keeps each thread free of any interruption and supports the scalability over any number of processors.

In Chapter 6, a novel approach for speeding up the generation of photo-realistic integral frames has been explained. The approach is named as integral reprojection. The idea of reprojection is to avoid tracing pixels and reuse results from the previous frame in order to generate the new frame. The chapter started by a brief description of the idea of projection that the points representing the scenes are reprojected onto the image plane. The mathematics of integral projection and the modelling of the cylindrical lens were described as well as the derivation of the novel integral projection matrix, which is used in the proposed camera model. The implementation of integral reprojection was described in detail showing the three stages of the reprojection process: application of

the normalising transformation, reprojection, and clipping against image window. The development of the reprojection algorithm is explained in terms of the problems faced through the integral reprojection approach and the strategies adopted to the solutions of the artefacts caused by: incorrect occlusion, reprojection on larger area, and missed pixels. Experiments show the big impact the algorithm has made on the execution time and the number of computations. The novel integral reprojection algorithm proposed accelerates integral ray tracing by four times more than its normal execution speed and saved 70% of the rays to be ray-traced in any frame regardless of the complexity of the scene.

Finally, Chapter 7 gives a summary of the research undertaken and the achievements made. Possible further developments of the current research work are also suggested.

Chapter 2

Overview of 3D Imaging Techniques

Traditional two-dimensional displays like cathode ray tubes (CRT) or liquid crystal displays (LCD) are popular for a wide range of applications. However, in current applications, these systems display the spatial information from only one perspective view. More and more visual imaging applications need to be able to portray natural and graphically generated environments in three dimensions. The greatly improved sensations of depth and naturalness provided by a 3D display can cause viewers to perceive an increase in the overall picture quality, leading them to prefer 3D presentation [Moto95, Taub02, 3dcgi].

A large variety of 3D imaging systems have been reported by a number of independent research groups, either as general-purpose display units or targeting a specific application [Sieg95, McCo95, Dodg97, Outw99]. Among them, autostereoscopic display, which provides 3D perception without the need for special glasses or headgear. This chapter gives a brief overview of 3D display technologies, concentrating on autostereoscopic (free-viewing) systems. An outline of the construction and operation

of the integral 3D imaging system developed by the 3D Imaging Group at De Montfort University (DMU) is then presented.

2.1 Three dimensional imaging systems

2.1.1 3D with glasses (stereoscopy)

3D displays that require the viewer to wear special glasses are reasonably well known. The earliest type of 3D display is stereoscopic imaging, which can be traced back to Wheatstone's work in 1832 [Valy66]. Figure 2.1 shows the Wheatstone stereoscope, in which two geometric two-dimensional (2D) drawings exhibiting disparity are viewed using a special device called a reflecting mirror stereoscope. Figure 2.2 is a top view of it. These primitive devices operated by displaying two images side-by-side, one for each eye. Therefore, the two views are channelled separately to the corresponding eye by the simple optical elements.

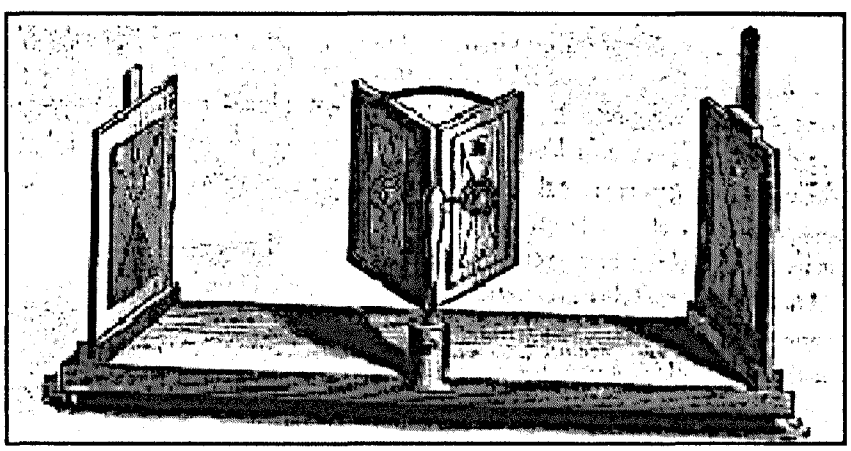


Figure 2.1: Wheatstone's stereoscope [Okos76].

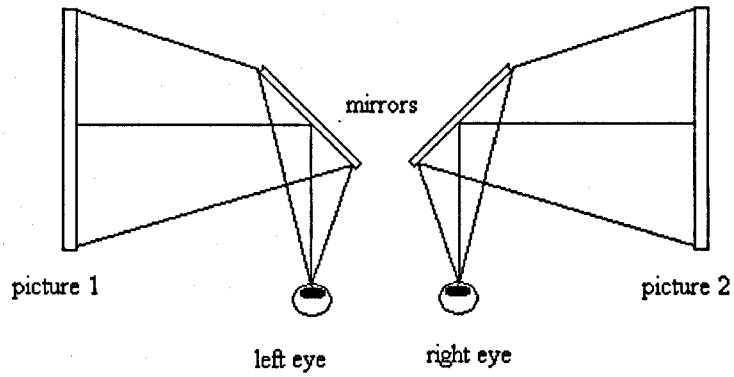


Figure 2.2: Top view of Wheatstone's stereoscope [Okos76].

In later developments of stereoscopic displays, viewers are required to wear special image selective glasses. The glasses themselves select which of the two images is visible to each of the viewer's eyes. The technology can be divided into three categories [Okos76, Bent80, Krat72, McAl93]:

I) Anaglyph method: The left-eye and right-eye images are projected or displayed using two different colours (red/green, cyan/magenta, etc). The viewer observes the images through a pair of colour glasses. The glasses act as a selective device so that each eye only sees one corresponding image.

II) Polarization method: The left-eye and right-eye images are projected or displayed through polaroid filters and are polarized orthogonally. The viewer observes the images through a pair of polaroid glasses so that each eye will perceive only one image.

III) Time division method: The two images are sent alternately to the two eyes using a double frame rate display combining with shuttered glasses. The shuttered glasses shield each eye alternatively and present the left-eye image to left eye and the right-eye image to the right eye.

Stereoscopy is one of the simplest ways by far to provide 3D perception using 2D display systems and many entertainment and commercial systems are using and

developing these technologies. The principle exists in using various channelling methods to separate left/right eye images to the corresponding eye. However, the various channelling methods impose their own disadvantages. For example, the anaglyph display, which presents the left and right components of a stereo pair simultaneously using colour coding, produces eye strain due to rivalry between each eye and suffers from colour rendition problems. Polarizing or time division shuttering methods produce similar eye strain as the light reaching the eyes is attenuated. Most of all, it is not comfortable for the viewer to wear a pair of glasses for long periods in order to see the 3D effect. This limits the acceptance and applications of the stereoscopic display.

2.1.2 Autostereoscopic displays

Autostereoscopic display systems (free-viewing) are more acceptable to observer and therefore are more commercially viable. Autostereoscopic systems can be subdivided into four types: Create the viewing of an object though reconstruction of the wave fronts, (Holography); Create viewing that occupies a true volume space (Volumetric Displays); Use complex tracking systems to channel the correct information to the viewer (Two-view Display, Head-tracking Image System and Multi-view Display System); Use a physical sampling optical device to encode the angular information contained in a scene (Integral Imaging System).

2.1.2.1 Holography

A hologram is a light wave interference pattern recorded on photographic film (or other suitable surface) that can produce a 3D image when illuminated by suitable light. The principle of holography was established by Denis Gabor [Gabo48, Gabo49]. Figure 2.3 illustrates how a hologram is recorded [Amateur Holography]. A coherent light source, laser beam, is required in the capture process in order to produce interference fringes on the recording medium. The laser beam is split into two beams. The reference beam is spread by a lens or curved mirror and aimed directly at the film plate. The object beam

is spread and aimed at the object. The object reflects light and the reflected light is incident on the holographic film-plate. The two beams (reference and reflected) then interact and form a recorded interference pattern on the film. During the replay process, a 3D image of the original object appears when the hologram is illuminated from the original direction of the reference beam. The produced 3D images are virtually indistinguishable from real objects. By shifting position, the viewer can look around or over objects in the foreground to see within the viewing angle what is behind them.

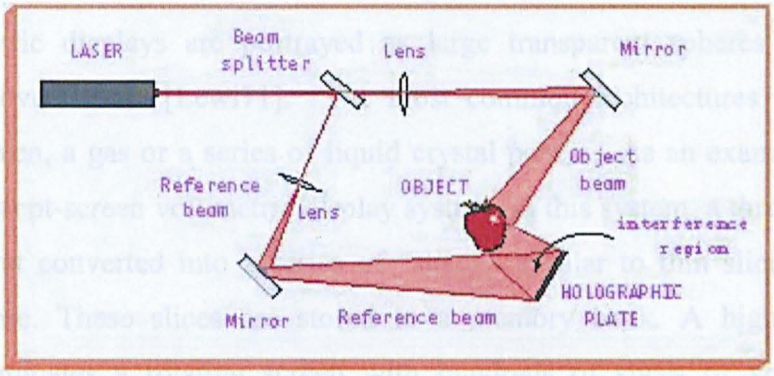


Figure 2.3: An illustration of holography recording [Amateur Holography].

The unique characteristic of holography is the idea of recording both the phase and the amplitude of the light waves from an object. Since all recording materials respond only to the intensity in the image, it is necessary to convert the phase information into variations of intensity. Holography does this by interfering coherent light from the object and the reference beams derived from the same source.

Many variations on and enhancements to the basic process have been proposed and demonstrated incorporating variations in both viewing conditions and fabrication techniques [Bent80, Kasp87, Outw99, Hari02]. Depending on the precise materials and technique used in the capture process, reconstruction may take place under coherent or incoherent illumination. The white light reflection hologram, which can be found on credit cards, magazine covers and recording packaging, etc, is perhaps one of the best

known developments from the earlier hologram. However, problems exist when attempting to transfer the holographic technique to the display of moving spatial images. An additional problem is the capture of natural scenes as there are the requirements in making hologram for coherent light sources, dark room conditions and high mechanical stability during recording. These considerations reduce the practical utility of the holography technique for general 3D spatial video imaging applications.

2.1.2.2 Volumetric displays

Most volumetric displays are portrayed as large transparent spheres with imagery seeming to hover inside [Lewi71]. The most common architectures use a rotating projection screen, a gas or a series of liquid crystal panels. As an example, figure 2.4 illustrates a swept-screen volumetric display system. In this system, a three-dimensional data set is first converted into a series of "slices," similar to thin slices of an apple around its core. These slices are stored in a memory bank. A high-speed digital projector illuminates a rotating screen with hundreds of slices of voxel data at a reasonable frequency. Viewers perceive a sharp 3D image due to the fusion caused by the persistence of vision in the eye.

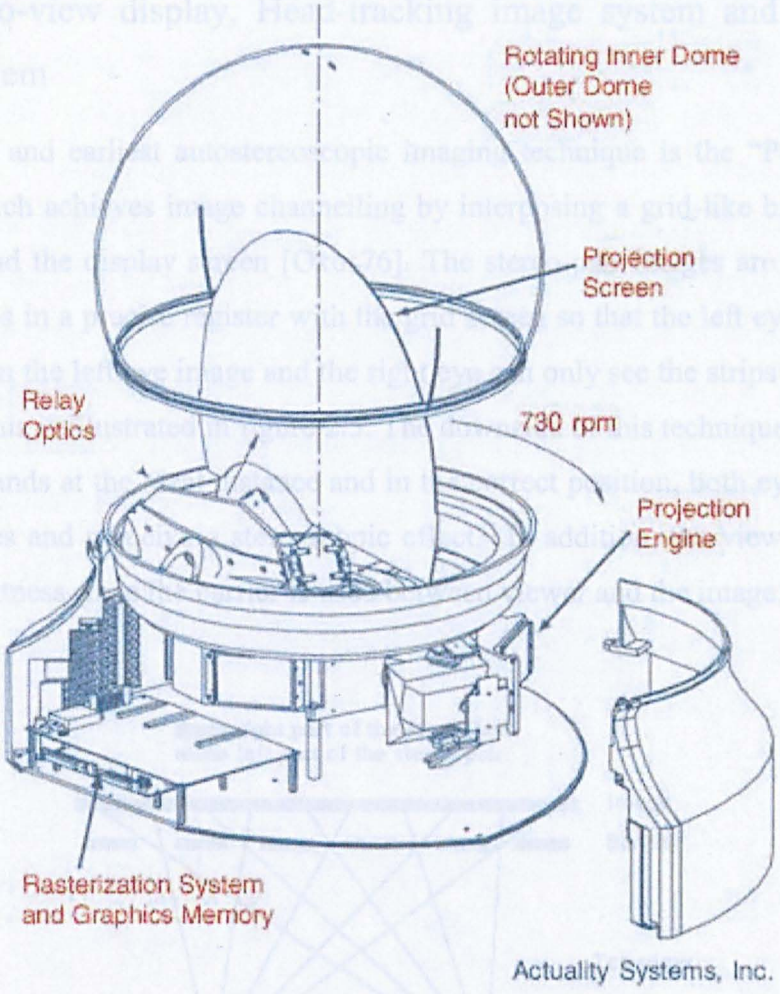


Figure 2.4: A swept-screen volumetric display system [Actuality Systems].

One benefit of volumetric displays is that they often have large fields of view, such as 360 degrees around the display, viewable simultaneously by an almost unlimited number of people. Volumetric displays are always difficult to design and make, which limits their application in the 3D display area.

2.1.2.3 Two-view display, Head-tracking image system and Multi-view display system

The simplest and earliest autostereoscopic imaging technique is the "Parallax barrier method", which achieves image channelling by interposing a grid-like barrier between the viewer and the display screen [Okos76]. The stereo-pair images are interleaved in alternate strips in a precise register with the grid screen so that the left eye can only see the strips from the left eye image and the right eye can only see the strips from the right eye image. This is illustrated in figure 2.5. The downside of this technique is that only if the viewer stands at the ideal distance and in the correct position, both eyes can see the correct images and perceive a stereoscopic effect. In addition, the viewer perceives a reduced brightness since the barrier is used between viewer and the image.

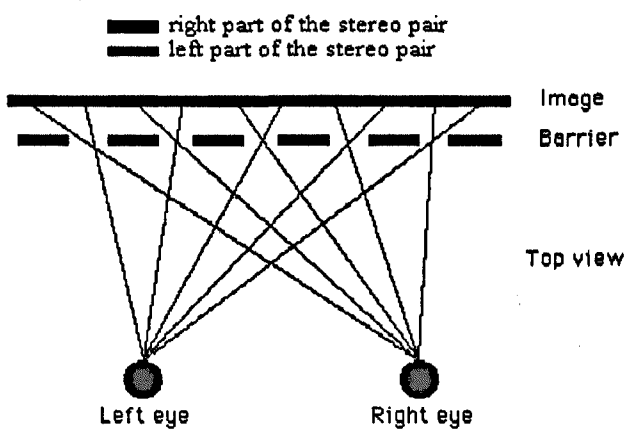


Figure 2.5: Viewing a parallax barrier display [Bour99].

If the position of the viewer's head is known, the appropriate images (left and right). could be displayed to the appropriate zones adaptively so that each eye can always see the correct images. This is the principle used in the head tracking systems [Tets94]. The limitation of most head tracking systems is that they are single-viewer since the display can only be adjusted to one viewer's eye position. This is only acceptable in some applications. Beside, it would be pointless to replace the wearing of special glasses with the wearing of a special head tracker for viewers to view the 3D effect.

A similar methodology is used in the lenticular screen method, where the lenticular lens is used as a directional selective screen to separate the left-eye and right-eye view and present the views to the correct eye. The simplest form of such a display is comprised of alternating vertical strips of the left-eye/right-eye images, the same as in parallax barrier display. See figure 2.6. However, each stereo pair of view strips is located precisely behind a lenticular screen during the display rather than the parallax barrier. The lenticular screen works in a similar way to the parallax barrier in directing the corresponding images to different eyes. The advantage over the barrier method is that refraction rather than occlusion is used hence better image brightness can be achieved from the lenticular screen display.

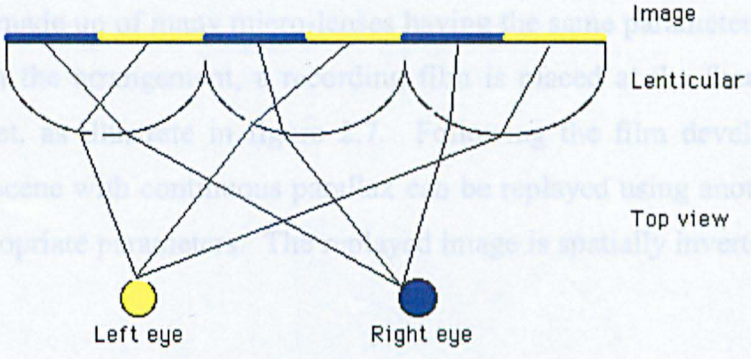


Figure 2.6: Viewing a lenticular screen display [Bour99].

To integrate a "look-around" capability and increase the number of viewing zones, multiple stereo viewpoints are provided by locating bands derived from many views [Harm96] [Dodg97], therefore allowing the viewer to move their head from side to side and see different aspects of the 3D scene. In recent years, many three dimensional television recording and display systems have concentrated on the method of using of multi-camera capture and multi-view displays to allow a certain degree of look-around capability. The viewing effect depends on the number of viewpoints. The difficulty of simultaneously generating or capturing enough views in real time at an affordable cost becomes the major problem in this type of system.

2.1.2.4 Integral imaging

Integral imaging is a technique that is capable of creating and encoding a true spatial optical model of the object scene in the form of a planar intensity distribution by using unique optical components. It is akin to holography in that 3D information is recorded on a 2D medium and can be replayed as a full 3D optical model. However, in contrast to holography, coherent light sources are not required for integral imaging. This conveniently allows more conventional live capture and display procedures to be adopted.

All integral imaging can be traced from the work of Gabriel Lippmann [Lipp08], where a micro-lens sheet was used to record the optical model of an object scene. The micro-lens sheet was made up of many micro-lenses having the same parameters and the same focal plane. In the arrangement, a recording film is placed at the focal plane of the micro-lens sheet, as illustrate in figure 2.7. Following the film development, a full natural colour scene with continuous parallax can be replayed using another micro-lens sheet with appropriate parameters. The replayed image is spatially inverted as shown in figure 2.8.

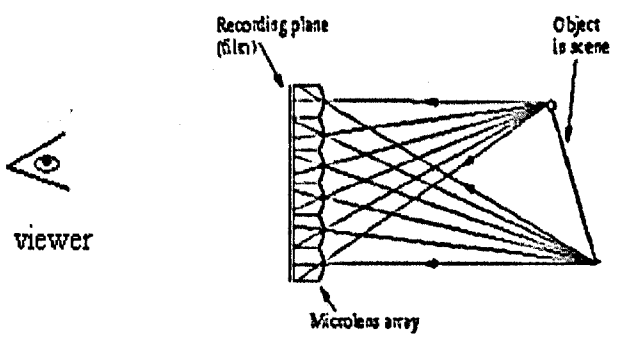


Figure 2.7: The recording of an integral image [Wu03].

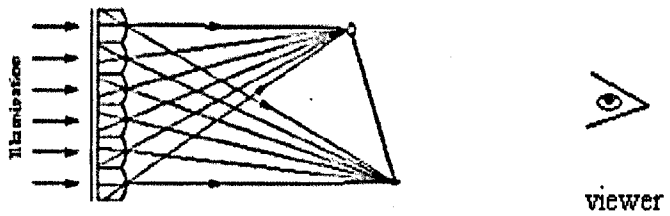


Figure 2.8: The replay of an integral image (The viewer perceives a spatial inverted 3D scene) [Wu03].

To overcome the problem imposed by the pseudoscopic (spatially inverted) nature of the integral image (II), a modification to the Lippmann system was proposed by Ives [Ives31], in which a second recording process is introduced before replaying, as shown in figure 2.9. When the second-stage photograph is replayed, a 3D image with correct spatial depth (orthoscopic) can be observed. See figure 2.10.

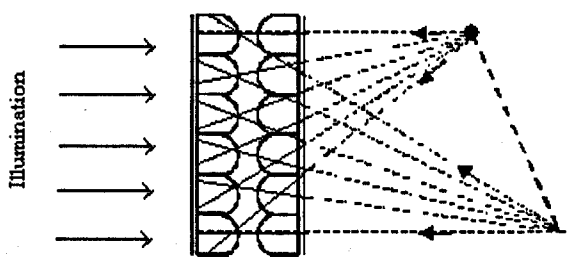


Figure2.9: A second stage recording of integral photograph [Wu03].

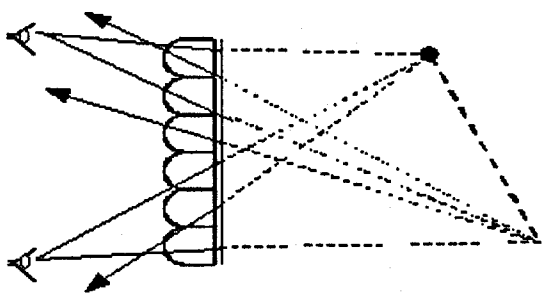


Figure 2.10: Replay and viewing of the orthoscopic image scene [Wu03].

The two-stage recording process can produce an orthoscopic 3D scene with corrected spatial position. However, substantial image quality degradation is introduced due to the distortions introduced by the micro-lenses and film emulsion, stray light, etc. To overcome this problem, a two-tier network as a combination of macro-lens arrays and micro-lens arrays was reported by Davies and McCormick in DMU [Davi94]. The two-tier network works as an optical "transmission inversion screen" which overcomes the image degradation caused by the two-stage recording process and allows direct spatially correct 3D image capture for orthoscopic replay. Theoretically, this network is able to capture object space from 0.3m to infinity. In consequence, the integral photographic technique pioneered by Lippmann has been improved. With recent progress in micro-lens manufacturing techniques, integral imaging is becoming a practical and prospective 3D display technology and hence is attracting much interest.

2.2 An advanced integral imaging system

The optics of an advanced form of integral imaging system employing a two-tier optical network was developed and has been described in detail by Davies and McCormick [Davi88, Davi94, McCo92, McCo94, McCo95]. The optical arrangement, shown in figure 2.11, comprises two macro-lens arrays placed equidistantly behind and in front of an auto-collimating transmission screen (ATS). The ATS is made up of two micro-lens arrays separated by their joint focal distance. The recording plane is a photographic plate whose position coincides with the focal plane of another micro-lens array.

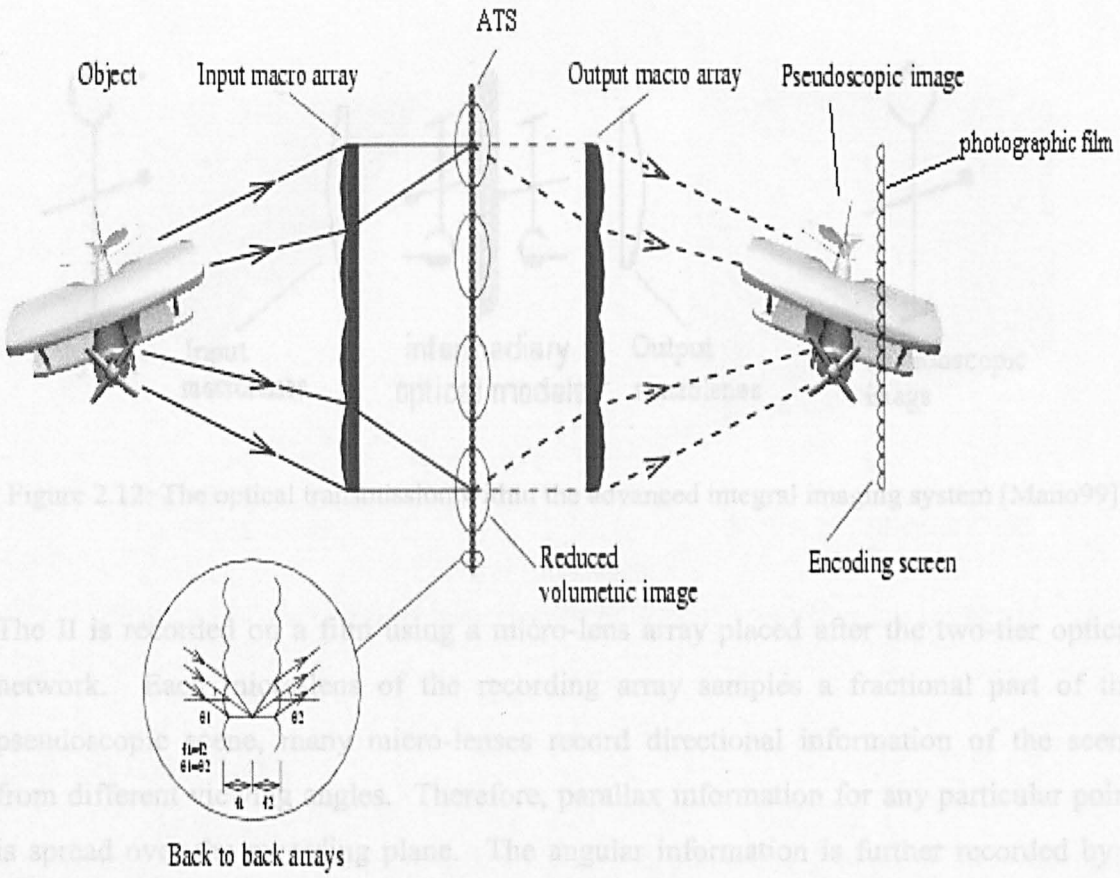


Figure 2.11: The advanced integral imaging system [Davi94].

The optical transmission process of the advanced optical system is illustrated in figure 2.12. The input macro-lens array first transmits the compressed object space to or near the central double micro-lens screen (ATS). The screen inverts the spatial sense of each intermediary image and simultaneously presents these spatially reversed 3D optical models to the corresponding output macro-lenses. The output macro-lens array then re-transposes the optical model to the correct spatial location. The final integrated optical model before recording, formed by the second macro-lens array, is a true 3D optical 1:1 reconstruction of the original object.

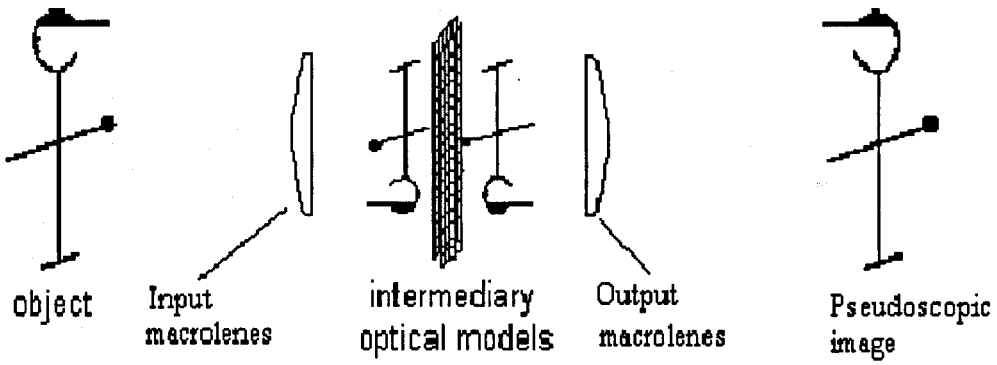
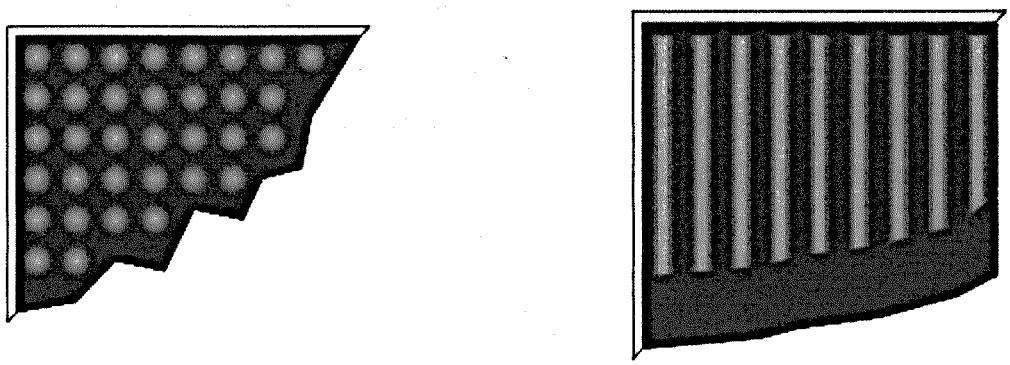


Figure 2.12: The optical transmission within the advanced integral imaging system [Mano99].

The II is recorded on a film using a micro-lens array placed after the two-tier optical network. Each micro-lens of the recording array samples a fractional part of the pseudoscopic scene, many micro-lenses record directional information of the scene from different viewing angles. Therefore, parallax information for any particular point is spread over the recording plane. The angular information is further recorded by a film placed at the focal plane of the micro-lens array. This recorded II can be replayed by overlaying it with a micro-lens array having the same parameters.

In the capture and display processes, micro-lens arrays are used in both the encoding and decoding of the planar intensity distribution. The arrays used for this purpose typically comprise square based spherical lens-lets, which are capable of encoding the object scene with continuous parallax in all directions. A section of such a lens array is illustrated in figure 2.13a. It is also possible to record and replay integral 3D images using lenticular sheets, which comprise many thin cylindrical lenses, as shown in Figure 2.13b. Integral images recorded in this way possess parallax only in one direction. It is worth mentioning that the integral image produced by the lenticular sheet is not the same as multi-view image with lenticular screen display. In the multi-view display system, the lenticular sheet is used merely for spatial de-multiplexing of multiple separate views of an object scene.



a)
Section of square based microlens array

b)
Section of lenticular sheet

Figure 2.13 Diagrammatic representation of the lens array ([Forman 1999](3))

2.3 Summary

3D imaging systems are the new generation of displays that give the viewer far more realistic perception than 2D displays.

In this chapter a brief description of 3D imaging system has been presented starting with Wheatstone's early work and his stereoscope. His stereoscope was one of the earliest models of stereoscopic displays that present a different view for each of the viewer's eyes. The concept developed to 3D glasses, which use one of three methods to present the left-eye image to the left eye and the right-eye image to the right eye. The first method is anaglyph method; the second method is polarization method; and the third method is time division method. Different 3D glasses share the disadvantages that they produce eye strain and the uncomfortable wearing of glasses for long periods.

Then the chapter illustrates the most important type of 3D displays, which are autostereoscopic displays. They give the viewer the chance to see 3D images without having eye strain or wearing glasses. The four main types of autostereoscopic displays have been described. The first type is holography, which uses a coherent light beam

(Laser) in the process of capturing 3D image of an object. The wave interference pattern is recorded on a photographic film or a recoding medium. There are many limitations on the viability of holography. The capture process of holograms requires coherent light sources, dark room conditions and high mechanical stability during recording. These considerations reduce the practical utility of the holography technique for general 3D spatial video imaging applications.

The second type of autostereoscopic displays is volumetric displays, which contain a high-speed digital projector illuminating a rotating screen with hundreds of slices of voxel data at a reasonable frequency. Viewers perceive a sharp 3D image at 360 degrees around the display, viewable from any direction. Volumetric displays face many difficulties in design and manufacture, which limited its application in 3D display area.

The third type of autostereoscopic displays channel the correct visual information to each of the viewer's eye. This includes systems like two-view Display, head-tracking image system, and multi-view display system. Two-view display use either parallax barrier or a lenticular lens to separate the left-eye and right-eye views to correct eyes. The backdrop of this technique that the viewer has to stand at the ideal distance and in the correct position in order to see the correct images. Head-tracking image system solves this problem by letting the viewer wears a head tracker device and sends the images to his position. Both systems limit the number of users to only one user. Multi-view displays provide Multi-stereo viewpoints by locating bands derived from many views therefore allowing the viewer to move their head from side to side and see different aspects of the 3D scene. The difficulty of simultaneously generating or capturing enough views in real time at an affordable cost becomes the major problem in this type of system.

The fourth type of autostereoscopic displays is integral imaging system. It provides encoding of a true volume spatial optical model of the object scene in the form of a

planar intensity distribution by using unique optical components. Unlike holography integral imaging does not need coherent light or special requirements for capturing or displaying. Integral imaging produces coherent spatial image that can be seen from different directions by cheap means without the need for wearing glasses or special devices. The history of integral imaging has been described and how that producing an orthogonal image was a problem in earlier integral imaging systems.

The advanced integral imaging system developed by Davis and McCormick has been illustrated. The system is able to produce an orthogonal image in a direct capture through a two-tier network. The two-tier network works as an optical "transmission inversion screen" which overcomes the image degradation caused by the two-stage recording process in earlier systems and allows direct spatially correct 3D image capture for orthoscopic replay.

The next chapter will describe one of the well known photo-realistic renderers which is ray tracing and how it has been modified in order to produce three-dimensional integral images.

Chapter 3

Overview of Ray Tracing

Photo-realistic image synthesis is described as: to create an image that evokes the same visual perception to a human observer as a real three-dimensional scene [Shir94, Che92]. In this field, the best lighting simulation algorithms to date are ray tracing [Whit80], radiosity [Gora84] and two-pass algorithm, which perform a radiosity pre-processing on the scene and use ray tracing to render the final image [Shir91, Kok94]. The difference between these algorithms is which light paths are approximated and which are correctly simulated. In this chapter the global illumination problem is addressed; different aspects of ray tracing algorithm are described as well as the migration of the ray tracing algorithm into integral imaging displays that produce three-dimensional photo-realistic computer-generated images.

Ray tracing is an approximated solution to the light transport problem, which is also known as the global illumination problem. The equations describing the physical problem of light transport in a three-dimensional environment are known. Solving these equations is the major theme in the rendering research. The next section explains the formulation of the global illumination problem.

3.1 Formulation of the Global Illumination Problem

The global illumination problem basically is a transport problem. Energy emitted by light sources is transported by means of reflections and refractions in a three-dimensional environment. Time dependent phenomena is not considered, which would result from considering the speed of light as a finite measure. Since the human eye is sensitive to radiance values, and considering images with photo-realistic quality, radiance values or average radiance values are computed over a certain area and solid angle. See Appendix. The latter means that fluxes should be computed for several areas of interest. These areas of interest are referred to as sets. The exact geometric shape of these sets can substantially vary, depending on the requested level of accuracy. Ray tracing algorithms define sets as surface points visible through a pixel, with respect to the aperture of the eye as shown in figure 3.1. Radiosity algorithms often define sets as surface patches with the reflecting hemisphere as the directional component as shown in figure 3.2.

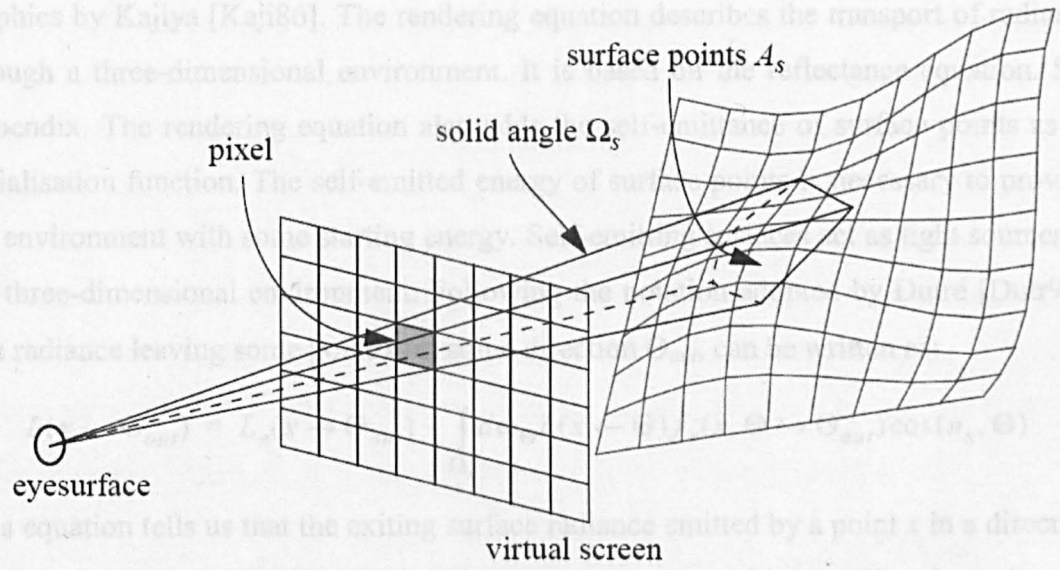


Figure 3.1 Set of surface points and directions corresponding to a pixel [Dutr96].

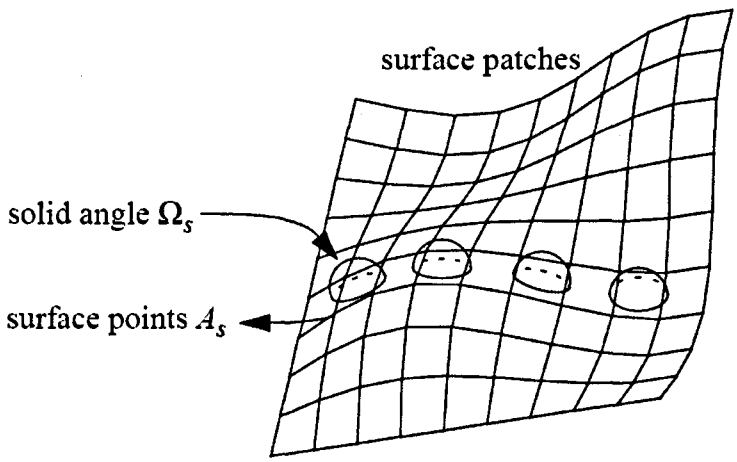


Figure 3.2 Sets for radiosity computations [Dutr96].

The fundamental transport equation used to describe the global illumination problem is called the rendering equation, and was first introduced into the field of computer graphics by Kajiya [Kaji86]. The rendering equation describes the transport of radiance through a three-dimensional environment. It is based on the reflectance equation. See Appendix. The rendering equation also adds the self-emittance of surface points as an initialisation function. The self-emitted energy of surface points is necessary to provide the environment with some starting energy. Self-emitting surfaces act as light sources in the three-dimensional environment. Following the notation adopted by Dutré [Dutr96], The radiance leaving some point x, in some direction Θ_{out}, can be written as:

$$L(x \rightarrow \Theta_{out}) = L_e(x \rightarrow \Theta_{out}) + \int_{\Omega_x} d\omega_\Theta L(x \leftarrow \Theta) f_r(x, \Theta \leftrightarrow \Theta_{out}) \cos(n_x, \Theta) \quad (3.1)$$

This equation tells us that the exiting surface radiance emitted by a point x in a direction Θ_{out} equals the self-emitted exiting radiance at that point and in that direction, plus any incident radiance from the illuminating hemisphere that is reflected at x in direction Θ_{out}. Figure 3.3 shows the diagram of the rendering equation. The material properties of surface point x are represented by the bi-directional reflection distribution function

(BRDF) $f_r(x, \Theta \leftrightarrow \Theta_{out})$, which returns the amount of radiance reflected into direction Θ_{out} as function of incident radiance from direction Θ.

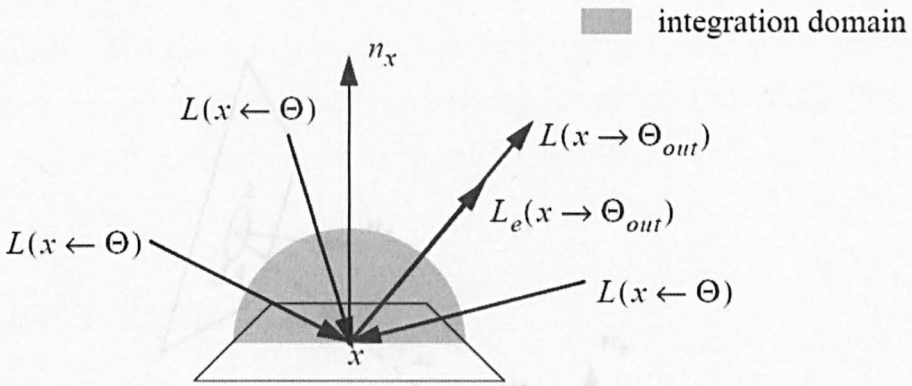


Figure 3.3 Transport of radiance: incident radiance is integrated over the hemisphere [Dutr96].

Emission can result from various physical processes, e.g. heat or chemical reactions. The emission can also be time-dependent for a single surface point and direction, as is the case with phosphorescence. Self-emitted radiance is merely considered as a function of position and direction. The rendering equation as given by equation (3.1) integrates all incident directions over the illuminating hemisphere. Sometimes it is more convenient to integrate over all visible surface points in the scene. By transforming the differential solid angles to differential surface areas, the equation can be rewritten, thereby integrating over all surface points in the scene. $d\omega_\Theta$ can be transformed to a differential area where dA_z is the closest point visible to x. See figure 3.4:

$$d\omega_\Theta = \frac{\cos(n_z, -\Theta) dA_z}{r_{xz}^2} \quad \text{and} \quad z = r(x, \Theta)$$

Substituting in equation (3.1):

$$L(x \rightarrow y) = L_e(x \rightarrow y) + \int_{A_x} dA_z L(x \leftarrow z) f_r(x, z \leftrightarrow y) \frac{\cos(n_x, \Theta)\cos(n_z, -\Theta)}{r_{xz}^2}$$

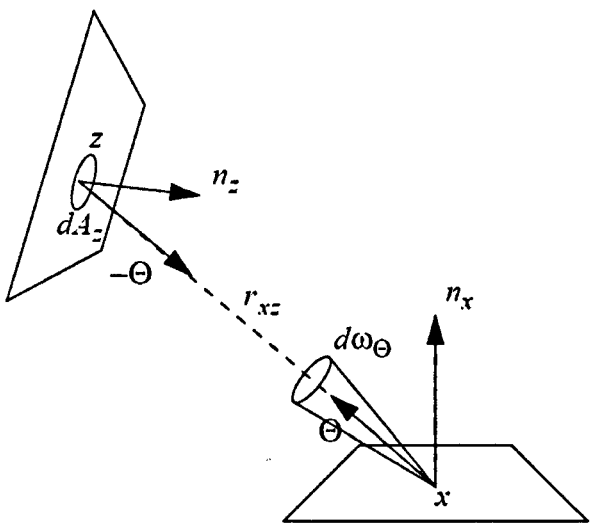


Figure 3.4 Transformation from hemisphere to surface integration [Dutr96].

where A_x denotes all surface points visible to x. By using the visibility function $V(x, z)$, the need for an integration domain A_x that is dependent on x, can be eliminated, and shift the visibility information from the integration domain towards the integrand.

$$L(x \rightarrow y) = L_e(x \rightarrow y) + \int_A dA_z L(x \leftarrow z) f_r(x, z \leftrightarrow y) G(x, z) \tag{3.2}$$

$$G(x, z) = \frac{\cos(n_x, \Theta)\cos(n_z, -\Theta) V(x, z)}{r_{xz}^2}$$

$G(x, z)$ is a geometric function, which expresses to what extent two surface points are able to exchange energy.

All the equations describing the transport of radiance are recursive integral equations with a fixed integration domain. Such equations are called Fredholm equations of the second kind. Generally, there is no analytical solution for these equations, and they have to be solved numerically [Dutr96]. The solution of these recursive integral equations for a large number of surface points and directions usually is the core of many existing global illumination algorithms such as ray tracing. In the next section, different aspects of ray tracing as a photo-realistic renderer will be described.

3.2 Ray tracing

All currently popular rendering algorithms approximate the rendering equation. The differences are in the type of error introduced by the different methods. One such approximation is ray tracing [Whit80]. The idea of the ray tracing is to simulate the light rays, which are reflected from objects seen by the eye (camera) in an opposite manner. It traces primary rays of light from the viewer's eye to the objects in the scene. This simple algorithm determines the colour and the intensity of a point at the closest intersection of a primary ray with an object. A centre of projection (the viewer's eye) and a window (image plane) on an arbitrary view plane are selected. The window may be thought of as being divided into a regular grid, whose elements correspond to pixels at a desired resolution. Then for each pixel in the window, a primary ray is spawned from the centre of projection through the centre of the pixel into the scene, as shown in figure 3.5. The colour of the pixel is set to that point of the object at the closest intersection. In order to model shadows, from the intersection point of the ray and the object, new rays are spawned towards each of the light sources. These rays, called shadow rays, are used to compute visibility between the intersection point and the light sources.

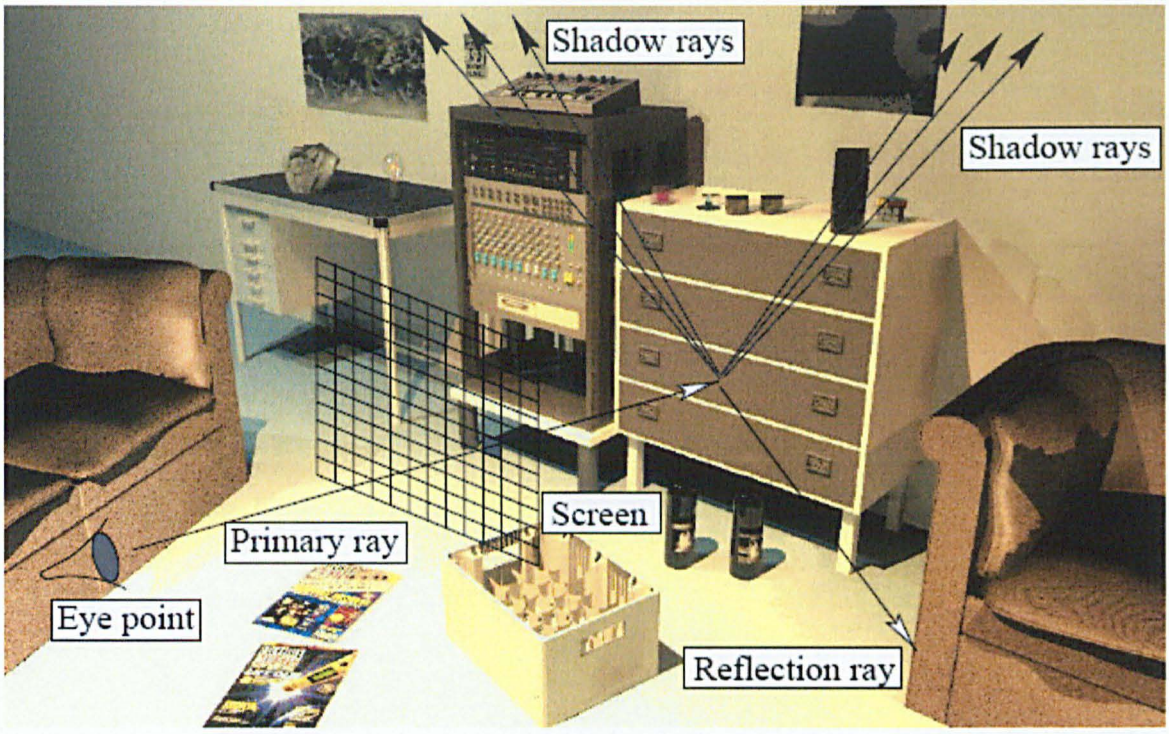


Figure 3.5: Overview of ray tracing.

Mirroring reflection and transparency may be modelled similarly by shooting new rays into the reflected and/or transmitted directions. These reflection and transparency rays are treated in exactly the same way as primary rays are. Hence, ray tracing is a recursive algorithm.

Tracing rays is a recursive process, which has to be carried out for each individual pixel separately. A typical image of 1000^2 pixels tends to cost at least a million primary rays and a multiple of that in the form of shadow rays and reflection and transparency rays. The most expensive parts of the algorithm are the visibility calculations. For each ray, the object that intersected the ray first, must be determined. To do this, a potentially large number of objects will have to be tested for intersection with each ray.

3.2.1 The virtual camera

The first stage in the ray tracing algorithm is the generation of primary rays, the rays that start the recursive ray tracing process. The camera model employed by most ray tracing systems generates primary rays by simulating a simple pinhole camera. In the pinhole camera model, a primary ray originates at the camera position, and is directed such that it pierces a point lying in the *image plane*. The image plane is subdivided into a two dimensional grid of pixels, which correspond directly to the pixels in the final output image. In a simple ray tracer, primary rays are spawned through the centre of each pixel in the image plane, and the resulting colour is stored in the corresponding pixel in the output image.

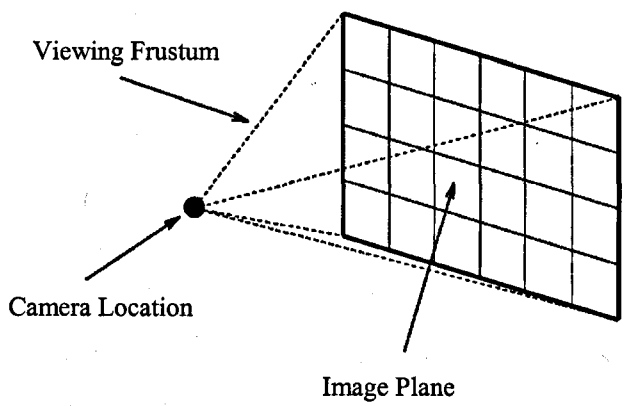


Figure 3.6. Virtual Camera and Image Plane [Ston98].

Figure 3.6 shows the camera location, image plane, and viewing frustum created by this simple model. In the simplest ray tracer implementations, only one primary ray is used to determine the colour of a pixel in the output image. Improved ray tracing algorithms spawn multiple primary rays through a given pixel's area in the image plane, in order to minimize aliasing artefacts. The number and direction of primary rays used to determine pixel color is determined by the antialiasing technique used by a given ray tracing system. Antialiasing is discussed in greater detail later in this chapter. The next step in the rendering process is visible surface determination.

3.2.2 Visible surface determination

Ray tracing differs from most other rendering techniques in the way it performs visible surface determination. Polygon oriented rendering techniques represent objects as collections of triangles or other planar polygons. In order to render curved geometry with adequate precision, polygon renderers tessellate curved surfaces with thousands of polygons. A significant problem with this approach is that no matter how finely a surface is tessellated it is always possible to choose a viewing configuration such that the edges of polygons can be seen. The only way to avoid this problem in a polygonal representation is to tessellate curves so that constituent polygons are smaller than a pixel in the resulting view. Many polygon based rendering systems use dynamic level of detail algorithms, which adaptively tessellate surfaces based on the viewing configuration.

Ray tracing performs visible surface determination separately for each pixel in the rendered image. Camera rays are intersected with objects in the scene. The intersection closest to the camera is taken as the visible surface. Since visible surface determination is done with at least pixel level resolution, curved surfaces are accurately sampled. Any kind of geometry that can be sampled via intersection tests can be represented in a ray tracing system. In order to render a sphere in a polygon based renderer, the sphere must be tessellated with polygons up to the required level of detail. A ray tracer tests a ray for intersection with the sphere. If an intersection exists, then the point or points at which the ray intersects the sphere are inserted into an intersection list. Since intersection tests are performed for separately for each pixel, the surface of the sphere is sampled with accurate resolution, yielding a curved looking surface regardless of the viewing configuration. The ray tracing process automatically produces the highest level of detail, since it samples geometric structure at each pixel. The results obtained by ray tracing are equivalent to tessellating a surface down to sub-pixel sized polygons. The geometric representation used in ray tracing can be much more memory efficient than that of a polygon renderer for this reason.

3.2.3 Surface shading

Once surface visibility has been determined through intersection tests with the objects in a scene, a colour must be assigned to the resulting surface at the closest point of intersection. Surface shading can be done many different ways. Ray tracing offers an elegant framework for performing a variety of shading operations. As with visible surface determination, ray tracing performs shading calculations for each pixel independently. Depending on the characteristics of an object's material, factors such as ambient light, diffuse light, reflection, refraction, and surface texture affect its colouring. The per-pixel evaluation of lighting and texturing used in ray tracing allows stunningly realistic images to be produced.

3.2.3.1 Ambient light

Although ray tracing systems are capable of realistically emulating many optical effects, they typically use simple algorithms for rendering shadows and indirect illumination. The standard ray tracing model only accounts for direct illumination; lighting attributed to rays, which travel from a light source directly to a surface, with no contributions from secondary reflections or transmissions from other objects. In the real world, light emanates from a source, possibly reflecting off of many surfaces before illuminating a surface. A good example of indirect lighting is a room with white walls, a table, and a lamp sitting on top of the table. In the real world, light emanates from the lamp on the table, and bounces around the room until it is completely absorbed by the surfaces in the room. Since the light is on the table, there is a region of shadow underneath the table. The table blocks all direct illumination from the lamp to the area underneath the table, leaving it in shadow. In the real world, the shadow cast by the table is not absolutely dark. Secondary reflections from the walls of the room illuminate the area underneath the table. The ray tracing algorithm accounts for direct illumination caused by the lamp on the table, but does not take into account the contributions of indirect lighting. Although the area under the table is not in complete shadow in real life, a ray traced image of this scene would yield an absolutely dark shadow under the table. A crude

approximation of the effects of indirect lighting may be obtained through the use of a constant ambient lighting value.

3.2.3.2 Shadows

The ray tracing algorithm handles shadows by spawning shadow test rays from the intersection point on a surface towards each of the lights in the scene. If a shadow ray encounters an object before it reaches the light, then the point is in the shadow of the occluding object. See figure 3.7. In a simple implementation, only one shadow ray is spawned for each light, and the transparency of an object is not taken into consideration. Improved shadow determination methods spawn multiple shadow rays towards each light, in order to reduce aliasing artefacts. More sophisticated ray tracers may test an occluding object's transparency, index of refraction, and colour in order to simulate filtering effects.

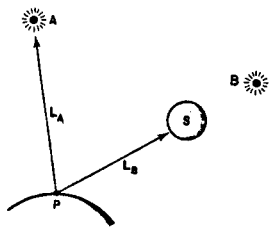


Figure 3.7: Light Source A contributes to point P's intensity but B doesn't [Glas98].

Shadow determination can be one of the most time consuming stages in the ray tracing algorithm, since a shadow ray is spawned for every light source in the scene, at every point where shading is performed. A simple optimisation for shadow testing is to provide an early exit in the intersection test code so that the shadow test terminates immediately when any intersection is found between the light source and the surface being shaded. This optimisation works for ray tracers that do not handle complex filtering. Two other optimisation techniques, which can greatly improve the efficiency of shadow testing, are the shadow cache and the light buffer algorithm [Hain86].

3.2.3.3 Diffuse shading

After shadow tests have determined that there are no occlusions blocking a light source, surface shading calculations for that light may begin. Diffuse shading determines the amount of light reflected from a dull matte surface such as typing paper. A simple method for approximating the diffuse reflection from a surface is to evaluate the cosine of the angle between the surface normal, and a vector pointing in the direction of the light. This can easily be evaluated by computing the dot product between the surface normal N and the light vector L, assuming both vectors have already been normalized. The resulting scalar value represents the diffuse contribution of the light source at that point on the surface. Figure 3.8 shows the vectors used in the diffuse shading calculations.

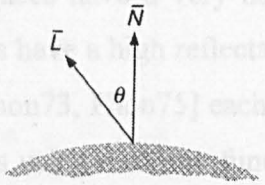


Figure 3.8. Diffuse Illumination of an Object [Fole90].

The diffuse illumination equation is:

$$I = I_p K_d \cos\theta \tag{3.3}$$

I is the computed intensity of a point on the surface of an object, I_p is the intensity of the illuminating light source, and K_d is the diffuse-illumination coefficient of the material. The illumination equation can be rewritten as:

$$I = I_p K_d (N.L) \tag{3.4}$$

Figure 3.9 shows a series of pictures of a sphere illuminated by a single light source using a diffuse-reflection model.

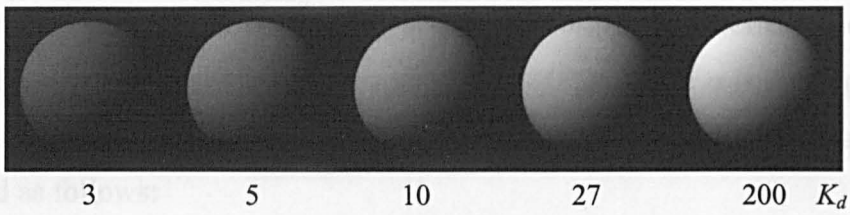


Figure 3.9: Diffuse illumination with different K_d [Fole90]

3.2.3.4 Specular highlights

Specular highlights simulate the reflective characteristics of shiny materials such as plastic and metal. These highlights are different from perfect specular reflection such as a mirror. Specular highlights are used to model imperfect reflecting surfaces. Rather than reflecting light only at one exact angle, imperfect surfaces reflect light over a range of angles. Smooth reflecting surfaces have a very narrow range of angles for which reflectance is high. Rough surfaces have a high reflectance over a wider range of angles. Romney [Romn69] and Phong [Phon73, Phon75] each developed shading equations for approximating specular reflections using a cosine function raised to an exponent. This technique provides a computationally inexpensive alternative to calculating specular reflections by spawning reflection rays. The drawback to this technique is that it does not take into account reflections of anything other than the lights in a scene. For mirror reflections or more accurate specular reflections, techniques that are more computationally expensive must be used. Figure 3.10 shows the geometry involved in performing shading for specular highlights based on the cosine exponent model.

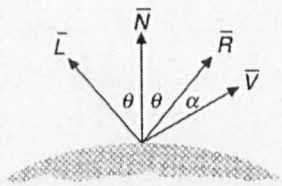


Figure 3.10: Phong highlighting [Fole90].

On a perfectly shiny surface such as a perfect mirror, light is reflected only in the direction of reflection R, which is L mirrored about N. The angle α is the angle between R and the direction to the view point V. The illumination equation for specular reflection is expressed as follows:

$$I = I_p K_s (R.V)^n \tag{3.5}$$

where K_s is the specular-illumination coefficient of the material and n is the material's specular-reflection exponent. Values of n typically vary from 1 to several hundred as shown in figure 3.11. For a perfect reflector, n would be infinite. Figure 3.12 shows pictures of spheres shaded using Phong illumination model with different values for k_s and n

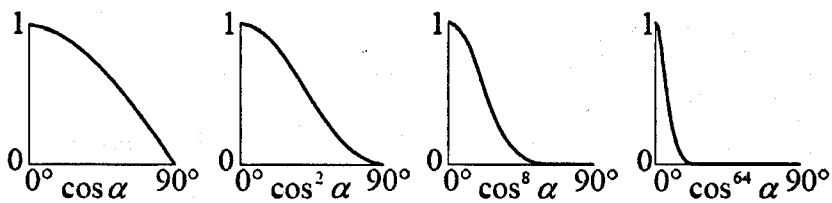


Figure 3.11: Different values of $\cos^n \alpha$ in the Phong illumination model [Glas89]

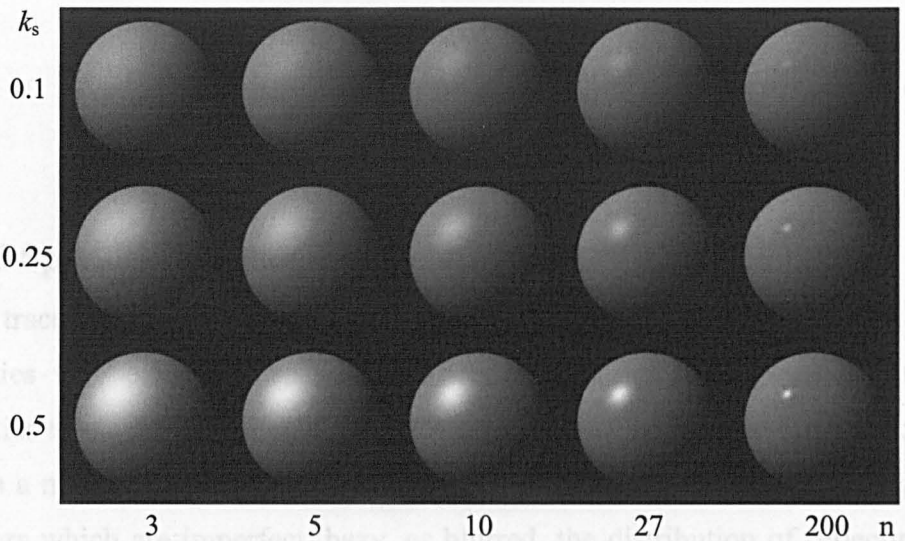


Figure 3.12: Spheres shaded using Phong illumination model with different values for k_s and n [Fole90].

The final illumination equation for a point's intensity consists of ambient model, diffuse model and specular reflection model:

$$I_\lambda = I_{a\lambda}K_a + f_{att}\, I_{p\lambda}[K_d\,(N.L) + K_s\,(R.V)^n] \tag{3.5}$$

where λ is the wave length, K_a is the ambient coefficient of the material, and f_{att} is an attenuation factor.

So far, monochromatic lights and surfaces have been described. Coloured lights and surfaces are commonly treated by writing separate equation for each component of the colour model. The use colour of an object is represented by one value of O_d for each component. For example the triple (O_{dR}, O_{dG}, O_{dB}) defines the object's diffuse red, green and blue components. In this case the illumination light is three primary components I_R, I_G, and I_B are reflected in proportion to K_dO_{dR}, K_dO_{dG} and K_dO_{dB} respectively. Specular highlight is affected by the properties of the surface itself and, in general, may have a different colour than the diffuse reflection. So the final illumination equation for the three primary components of point intensity is:

$$
\begin{aligned}
I_R &= I_{aR}K_aO_{dR} + f_{att}\, I_{pR}[K_dO_{dR}(N.L) + K_sO_{sR}(R.V)^n] \\
I_G &= I_{aG}K_aO_{dG} + f_{att}\, I_{pG}[K_dO_{dG}(N.L) + K_sO_{sG}(R.V)^n] \\
I_B &= I_{aB}K_aO_{dB} + f_{att}\, I_{pB}[K_dO_{dB}(N.L) + K_sO_{sB}(R.V)^n]
\end{aligned}
\tag{3.6}
$$

3.2.3.5 Specular reflection

A ray tracer can simulate specular reflections for surfaces, which have mirror-like properties very easily, by spawning reflection rays and adding the resulting colour intensities to the intensities calculated for other surface properties. For perfect reflectors such as a mirror, the distribution of reflection rays lies entirely along a single ray. For reflectors which are imperfect, hazy, or blurred, the distribution of reflection rays lies over a range of angles, which are narrower with nearly perfect reflectors, and wider for hazy or blurred reflectors. Figure 3.13 shows surface incident rays bouncing off of a planar mirror, hitting another object.

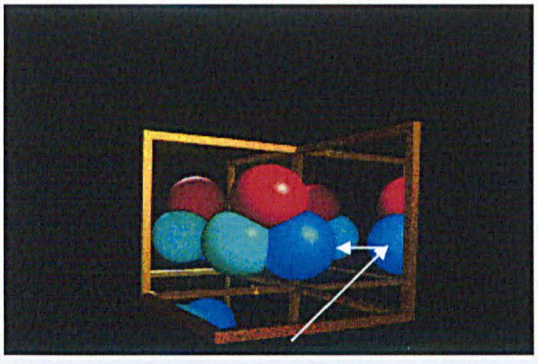


Figure 3.13: Specular reflection [HyperGraph].

3.2.3.6 Refraction

All of the shading techniques discussed up to this point have dealt with opaque surface characteristics. Transparent objects may allow transmission of light through the volume of the object depending on the index of refraction for the material of the object and the index of refraction of surrounding space. In a simple implementation, refractive effects are ignored, and a transmission ray continues through the material along the same path

as it enters and leaves an object. Refractive effects can be calculated using Snell's law. Figure 3.14 shows refraction at the interface between two transparent materials with different indices of refraction.

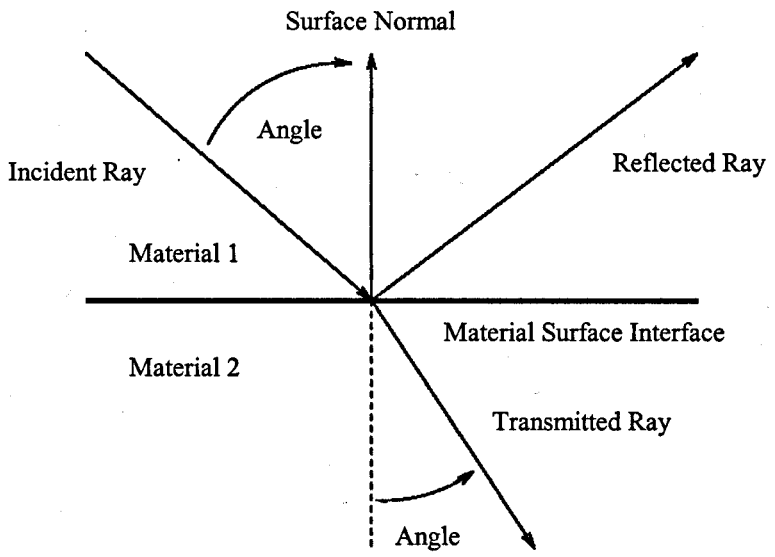


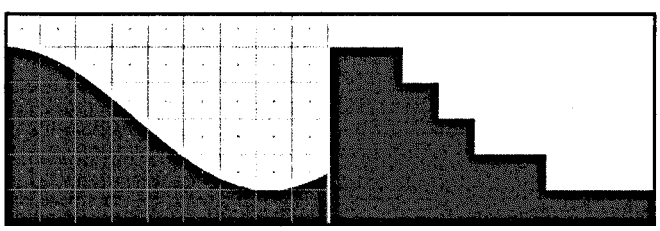
Figure 3.14: Refraction [Ston98]

A significant difficulty in implementing refractive effects properly lies in the task of keeping track of the indices of refraction for the space a ray is entering or leaving at any given time. When a new ray is created, it starts in some material. As the ray continues through space, it enters and leaves other materials, which may have different indices of refraction. For ray tracing systems that only implement solid geometric primitives, the task is straightforward. Indices of refraction can be determined by the solids the ray is entering and leaving. For ray tracing systems, which allow the use of polygonal meshes for the representation of solids, keeping track of the correct indices of refraction can be difficult. In these systems, some other method must be employed to help the ray tracer recall what indices of refraction to use on the two sides of the interface between the materials. One possible solution to this problem is the use of a stack to keep track of the index of refraction for the space the ray is leaving and entering at each interface. When the ray passes into an object, the index of refraction for that object is pushed onto the

top of the stack. When the ray leaves the object, the index of refraction is popped off of the stack. Some problems appear in resulted images rendered by ray tracing because of the approximations of the light behaviours. One of them is aliasing.

3.2.4 Aliasing

Ray tracing is based on discrete sampling of a continuous sample space. Image quality obtained by ray tracing is the direct result of the quality and of the samples taken. In figure 3.15, the picture on the left shows the sampling grid superimposed on the original scene. The picture on the right is the rendered image. A jagged profile is quite evident in the rendered image. Also known as *jaggies* or *staircasing*. Jaggies are an instance of a phenomenon known as *aliasing*. . In order to achieve the highest image quality, *antialiasing* techniques may greatly improve image quality with some additional computational cost.

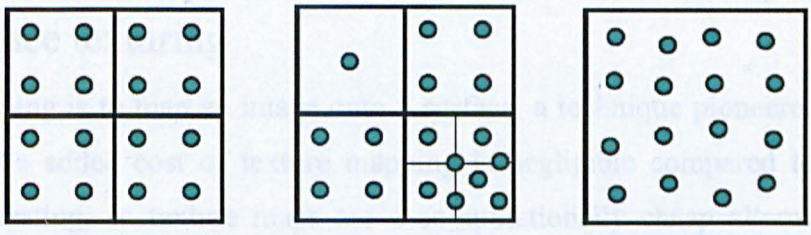


Original Rendered

Figure 3.15: Jaggies [HyperGraph].

Antialiasing algorithms typically cause multiple primary rays to be spawned for each output pixel. The number of extra rays and their directions are determined by the antialiasing algorithm. Since aliasing can occur both spatially and temporally, advanced ray tracers used for animation may distribute rays temporally as well as spatially. A variety of approaches to antialiasing may be used. Supersampling is a very simple antialiasing technique which subdivides each pixel into smaller subpixels along a uniform grid. Primary rays are then spawned for each of these subpixels, and the

resulting colours are combined using simple averaging, or a weighted average, to produce the resulting pixel. Although easy to implement, supersampling is still quite susceptible to aliasing effects, and can be computationally wasteful when overused. A slight modification to the supersampling technique can make it more computationally efficient, and give it greater flexibility. Adaptive supersampling attempts to determine areas where aliasing effects occur, and automatically subdivides a pixel into subpixels, and potentially recursively subdividing into even smaller subpixels based on differences between adjacent pixels or subpixels. Adaptive supersampling techniques generally employ simple metrics, which measure the variance between the colours of neighbouring pixels. If neighbouring pixels differ by more than a certain threshold, they are recursively supersampled until either they differ by less than the threshold, or a maximum cut-off subdivision depth has been reached. The resulting pixels are then averaged or filtered to produce the final pixel colour stored in the resulting image. Adaptive supersampling based on a uniform grid subdivision scheme can still suffer from grid alignment aliasing, even when supersampled to great depth. In order to combat this problem, samples may be taken in a non-grid-aligned fashion. The sample points may be determined pseudo-randomly for stochastic sampling, or they may be chosen based on a statistical distribution. These techniques avoid grid alignment aliasing artefacts and may require fewer samples to produce results of equal quality to those obtained by grid-aligned sampling techniques of greater depth. See figure 3.16.



(a) (b) (c)

Figure 3.16: (a) Supersampling [HyperGraph]
(b) Adaptive supersampling (c) Stochastic sampling.

3.2.5 Numerical precision and recursion

The ray tracing algorithm can encounter problems when rendering some scenes due to the finite precision of floating point arithmetic on computers. This problem can manifest itself in the generation of rays for reflection, refraction, and shadows. The problem occurs when a new ray is generated at a hit point on an object, going in some new direction. In some cases a ray will tend to intersect the same surface it is supposed to be leaving. In order to avoid this problem, a ray tracer can manipulate the starting point of the ray so that it is far enough off of the surface it is leaving that the intersection testing routines do not erroneously intersect with the surface the ray is leaving. Another problem that can occur is the generation of an infinite, or nearly infinite number of reflection or transmission rays. In a scene containing two perfectly parallel, perfectly reflective mirrors, it is possible for a ray, which is perfectly perpendicular to endlessly generate reflection rays. In the case of transmission, a ray may encounter total internal reflection, inside of an appropriately curved object such that it also endlessly generates transmission rays. One way to prevent a ray tracer from getting stuck in such pathological cases is to set a hard limit on the maximum level of recursion for the rays in a scene. A recursion limit of one would only allow primary rays to be traced. Recursion levels greater than one would allow reflections and transmissions to be ray traced. When the maximum level of recursion is reached, rather than generating reflection or transmission rays, the scene background color is used [Glas98].

3.2.6 Surface texturing

Texture mapping is to map an image onto a surface, a technique pioneered by Catmull [Catm74]. The added cost of texture mapping is negligible compared to the cost of intersection testing, so texture maps are a computationally cheap alternative to high geometric complexity. Texture maps can provide levels of detail, which are impractical to represent with highly tessellated surfaces of constant textures.

3.2.6.1 Coordinate transformation

In order to implement texture mapping in a ray tracer, the first step is to create a mapping function, which takes a 3D coordinate as its input, and generates a 2D parametric texture coordinate as its output. These transformations provide the mechanism for mapping a two dimensional image or procedural texture, onto a three dimensional surface. An easy coordinate transformation to implement is a cylindrical mapping. For a cylindrical mapping, the polar texture coordinate u could be used in order to index the x coordinate of the mapped image, and the coordinate v (along the axis of the cylinder) could be used in order to index the y coordinate of the mapped image. The polar u and v coordinates are derived by transforming world coordinates into coordinate system of the cylinder and then performing a standard polar coordinate conversion. See figure 3.17.

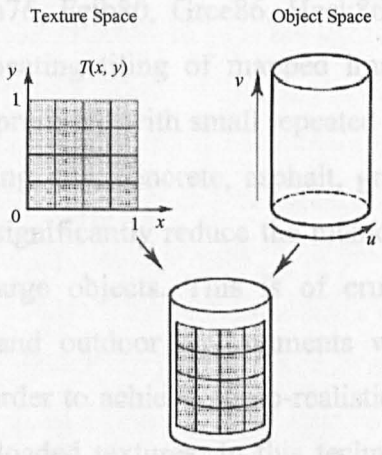
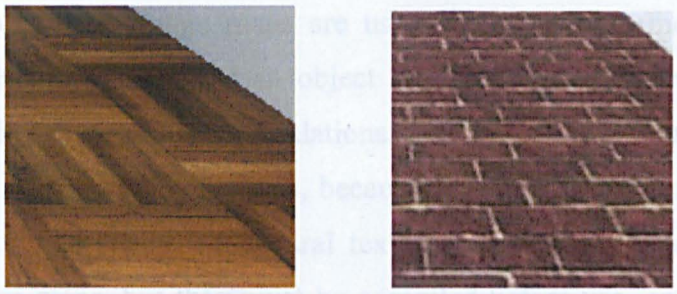


Figure 3.17: Texture mapping on a cylinder [MPI].

3.2.6.2 Texture lookup

Once the ray tracer has mapped a 3D object space coordinate to its corresponding 2D texture space coordinate, texture mapping routines generate a corresponding colour that will be used to paint the hit point on the surface of the object being textured. Since the texture space is two dimensional, this is a relatively simple procedure. This is the stage

where procedural textures and image mapped textures diverge. In the case of procedural textures, a texturing procedure is called, which calculates the mapped color based upon an equation, algorithm, or other rules. Procedural texturing functions may be arbitrarily complex, potentially calling many additional subroutines while generating texture color. For image mapped textures, u and v texture coordinates are used as indices into a two dimensional array containing the actual pixel data for the mapped image. In order to make image mappings more flexible, scaling, rotation, and translation operations may be applied to the 2D texture coordinates to alter the size and location of an image map on the surface of the target object. Excessive image map scaling, or insufficient image map resolution will result in aliasing artefacts in the resulting image [Watt93]. In order to combat this problem detailed images should be used, or their mappings should be scaled to a relatively small size in world coordinates. This is the most basic of image mapping techniques. Higher quality results are possible through the use of interpolation [Heck91] and filtering [Blin76, Feib80, Gree86, Heck86]. Modulo functions may be used to make endlessly repeating tiling of mapped images in texture space. Many textures can be accurately reproduced with small repeated images. Examples of textures include brick, parquet flooring, tile, concrete, asphalt, grass and meadow, and others. See figure 3.18. Tiling can significantly reduce the memory requirements for mapping high detail textures onto large objects. This is of crucial importance for realistic rendering of large indoor and outdoor environments where thousands of different textures may be needed in order to achieve photo-realistic results. Other optimizations involve the use of demand-loaded textures. In this technique, textures are not loaded until they are first referenced during the rendering process. Additionally, texture caches may be used, so that extremely complex environments may be rendered, including scenery, which uses more texture than, can be held in the physical memory of a computer. A texture caching and paging system can be used to unload textures which are only used for part of a scene, based on a least recently used paging scheme.



(a) (b)

Figure 3.18: Repeated tiling (a) Parquet (b) Bricks [HyperGraph].

3.2.6.3 Improving mapping quality

Since the ray tracing process results in point sampling of texture space, antialiasing techniques for texture mapping attempt to provide a more accurate determination of colour than would be obtained by point sampling itself. One way to reduce aliasing effects when doing image mapping is to use interpolation [Watk93] during the lookup of the texture coordinate and its image map pixel. Instead of returning the colour at the nearest image pixel, we can interpolate between the 4 nearest pixels, to come up with a more accurate, and smoother image. This technique adds very little additional overhead, but makes a big difference in the visual quality of scenes where the camera focuses on a mapped object at a high level of magnification. Images used in texture mapping for animation should be chosen with care, especially when used with tiling. When images are tiled over a surface, any discontinuities at the tile edges are easily visible in the final image. Tiling artefacts may be avoided by performing edge blending on image maps prior to rendering.

3.2.6.4 Image mapped textures

Image mapping is a technique for applying textures onto the surfaces of objects based on a world coordinate, texture coordinate mapping, and a source image. This technique uses the same kind of mapping transforms that procedural textures use but has its own

advantages and problems. Image maps are used whenever it is more convenient to simulate the surface appearance of an object by colouring it according to an image rather than by using mathematical calculations. See figure 3.19. Image maps are well suited for general rendering applications, because they are easily created with drawing programs and by photography. Procedural texture maps [Eber94] are typically more flexible than image maps, but they must be compiled before use, or interpreted at run time, both of which have their problems. Image maps typically require much more memory at run-time than procedural textures, so they are impractical when memory is limited. In order to apply images onto the surfaces of objects, we must establish a mapping function between the three dimensional world coordinate system and the coordinate system used by the texturing algorithm. Most texture spaces are two dimensional, with few exceptions. Although textures are mostly two dimensional, this does not limit their utility, the 2D texture space can be applied to a 3D geometric space in many different ways. Some common examples of mappings are spherical, cylindrical, rectangular-planar, and polar-planar. Each of these types of mappings, takes a 3D geometric input coordinate, and outputs a 2D texture coordinate in (u, v) parametric space of the mapping.

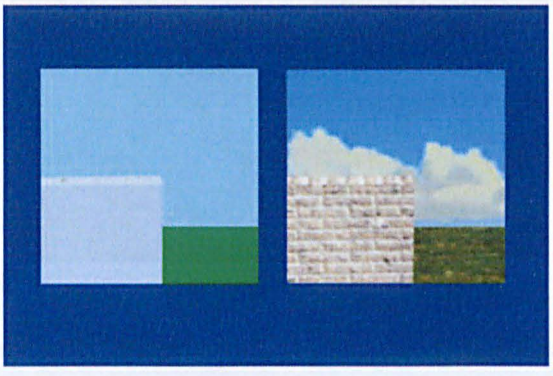


Figure 3.19: Image mapped texture [HyperGraph].

3.2.6.5 Procedural texturing

Procedural texturing is similar in many respects to image mapped texturing. Instead of determining shading parameters through referencing an image, a procedural texture generates shading parameters by executing procedural texturing algorithms. Procedural textures can generate highly complex textures with very little memory usage. Procedural textures typically generate detail on the fly rather than by pre-calculating large tables. Procedural textures and image mapped textures may be combined to provide highly complex texturing combinations. Procedural textures can be organised in hierarchical texture trees, higher levels of the texture tree combine multiple lower level textures. An example of hierarchical texturing is a chess board. A two dimensional checker pattern can be built from a checkering procedure and two lower level textures. The lower level textures could be image mapped textures or they could be procedural textures. Using this technique it is easy to create arbitrarily detailed textures. See figure 3.20. Next section describes the generation of photo-realistic integral images using ray tracing.

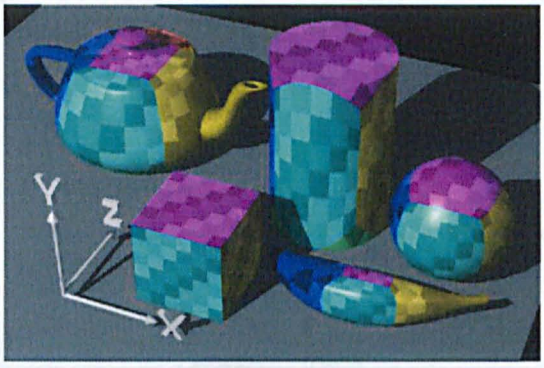


Figure 3.20: Procedural mapped texture [HyperGraph].

3.3 Computer generation of integral images using ray tracing

All the described quality of ray tracing can be migrated to three-dimensional integral imaging and opens the door to computer graphics application to facilitate integral imaging displays. Integral imaging increases the perception of the photo-realism in ray tracing for the observer. Computer generation of integral images has been reported in several literatures [Min01, Hall97, Naem01]. Computer generation of integral images is very useful where an II can be replayed using a LCD monitor by overlaying it with a lenticular sheet. Cartwright [Cart99] modelled the optical system of integral imaging and applied it inside a ray tracing renderer. This new renderer is represented in the thesis by the term integral ray tracing (IRT). For both lenticular sheets and micro-lens arrays, each cylindrical lens or micro-lens lens acts as a separate camera. The virtual scene straddles the modelled lenticular sheet as well as the image plane as shown in figure 3.21. Each camera spawns primary rays through the pixels that lay underneath it. Forefront intersection-points, which are closer to the viewer, are recorded into the image.

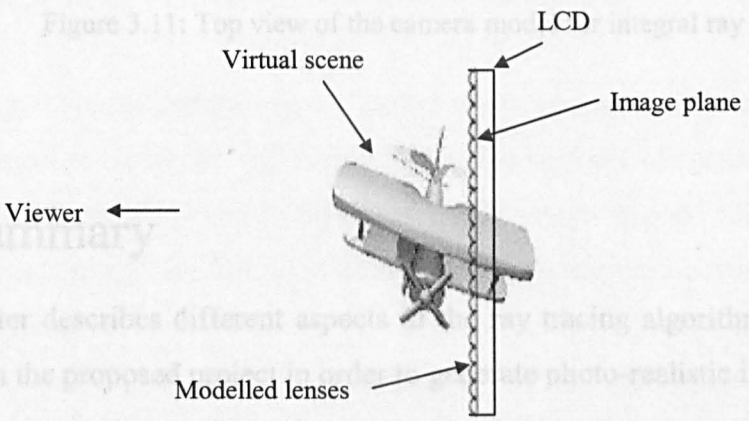


Figure 3.21: Top view of the modelled optical system of integral imaging.

3.3.1 Camera model

The used camera model is the pinhole approximation for each micro-lens. For a cylindrical lens an axial model is used. Each lens acts like a separate camera that has two-way recording directions. The result is effectively a multiple camera. Each micro-lens or cylindrical lens records a sub-image of the virtual scene from a different angle. See figure 3.11. Primary rays pass through the centre of the micro-lens and the image plane. The scene image straddles the micro-lens array. Therefore there are two recording directions, in front and behind the micro-lens array.

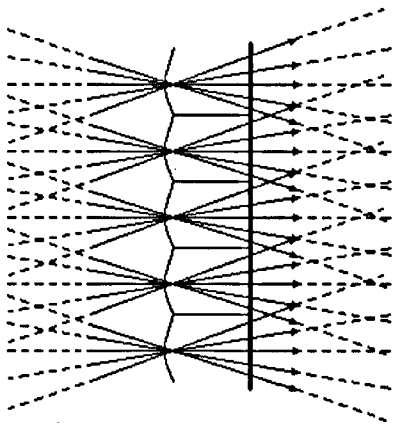


Figure 3.11: Top view of the camera model for integral ray tracing

3.4 Summary

This chapter describes different aspects of the ray tracing algorithm which is the used renderer in the proposed project in order to generate photo-realistic integral images.

Ray tracing is a light simulator and is one of approximated solutions to the global illumination problem. The introduced global illumination problem is a light transport problem. Energy emitted by light sources is transported by means of reflections and refractions in three-dimensional environment. The used fundamental transport equation

in order to describe the global illumination problem is called the rendering equation. Since the human eye is sensitive to radiance values. Radiance values are computed over a certain area and solid angle. All equations that are describing the transport of radiance are recursive integral equations with a fixed integration domain and have to be solved numerically.

Ray tracing simulates light rays, which are reflected from objects seen by the eye (camera) in an opposite manner. It traces rays from the viewer's eye to the light sources in the scene. This simple algorithm determines the colour and the intensity of a point at the closest intersection of a primary ray with an object. Ray tracing performs visible surface determination separately for each pixel in the rendered image. This property assures that curved surfaces are accurately sampled.

Ray tracing also offers an elegant framework for performing a variety of shading operations. Depending on the characteristics of an object's material, factors such as ambient light, diffuse light, reflection, and refraction affect its colouring. The per-pixel evaluation of lighting and texturing used in ray tracing allows stunningly photo-realistic images. Ray tracing can also handle shadows by spawning shadow rays from the intersection point on a surface towards each of the lights in the scene.

Ray tracing supports texture maps. Texture maps can provide levels of details, which are impractical to represent with highly tessellated surfaces of constant texture. They are two main categories in texture mapping. First is image mapped textures and second is procedural texturing. The former is used wherever it is more convenient to simulate the surface appearance of an object by colouring it according to an image rather than by using mathematical calculations. Procedural textures can generate highly complex textures with very little memory usage. Procedural textures and image mapped textures may be combined to provide highly complex texturing combinations. All the described quality of ray tracing can be migrated to three-dimensional integral imaging and open the door to computer graphics application to facilitate integral imaging displays. That will increase the perception of the photo-realism for the observer.

Computer generation of integral imaging using ray tracing is described with its camera model. The new camera model is a two-way pinhole model for each of the micro-lenses and a two-way axial model for cylindrical lenses. This model allows the scene to straddle the micro-lens array or the lenticular sheet in order to achieve three-dimensional out-of-screen image. Each lens is modelled as a separate camera that generates a sub-image containing a different view from its neighbour lens.

Computer-generated integral image can be replayed using LCD by overlaying it with a lenticular sheet or a micro-lens array. This system allows all graphical applications to benefits the three-dimension with the use of a cheap lenticular sheet.

Next chapter describes the traditional acceleration techniques that are used in order to speed up ray tracing algorithm.

Chapter 4

Traditional Acceleration Techniques for Ray Tracing

One of the greatest challenges for ray tracing is the efficient execution. Ray tracing is always claimed as being too computational. Efficiency is therefore a critical issue and has the focus of much research from the beginning. This had led to approaches involving data structures and numerical statistical methods. Because integral ray tracing is an adapted version of the original ray tracing algorithm, traditional ray tracing acceleration techniques can also be used in integral ray tracing.

The generality of ray tracing is due to its almost exclusive dependence upon a single operation: calculating the point of intersection between a ray in three-space and a geometrical entity or a primitive object. Examples of primitive objects include elementary shapes such as polygons, spheres and cylinders as well as more complex shapes such as parametric surfaces and swept surfaces. The major concern is the intersections of each ray with a large collection of primitive objects defining a scene. This ultimately is reduced to the computation of the point of intersecting closest to the ray origin, which results from any of the individual primitive objects in the scene. The cost of this operation typically overshadows everything else, accounting for the vast bulk of time consumed by ray tracing. A statistic reported by Whitted [Whit80] suggests

that more than 95% of the time can be spent performing this operation for complex scenes.

In this chapter, Glassner classifications of the accelerating techniques [Glas89] are described as well as some of their famous algorithms.

The intersection problem has a trivial but usually impractical solution, which is commonly referred as *exhaustive ray tracing*. This solution implies intersecting each ray with the scene by simply testing each and every primitive object and retaining the nearest point of intersection if one exists. This has a time complexity that is linear with the number of objects.

4.1 A board classification

Facing the task of accelerating the process of ray tracing, Glassner [Glas89] introduced three strategies to consider: (1) reducing the average cost of intersecting a ray with scene, (2) reducing the total number of rays intersected with the scene, and (3) replacing individual rays with a more general entity. These appear in figure 4.1 as "faster intersections", "fewer rays", and "generalised rays" respectively. In the following sections of this chapter, these strategies and their different techniques will be discussed.

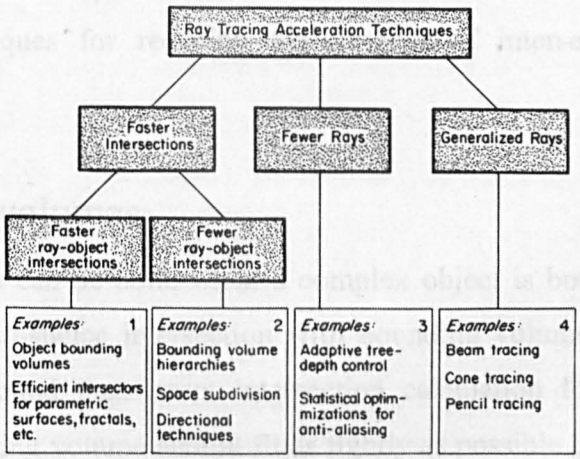


Figure 4.1: A broad classification of acceleration techniques [Glas89].

4.2 Faster intersections

The category of faster intersections further separates into the subcategories of "faster" and "fewer" ray-object intersections. The former consists of efficient algorithms for intersecting rays with specific primitive objects, while the latter addresses the larger problem of intersecting a ray with an environment using a minimum of ray-object intersection tests. The rest of this section will discuss some of the algorithms for both subcategories.

4.2.1 Efficient intersection tests

Intersection tests can be made faster by employing algorithms, which take advantage of the unique properties of geometric primitives. An intersection test with a sphere can be done a number of ways [Ston98]. Mathematically, a sphere is a quadric surface, and intersection tests can be performed by solving for an intersection between a ray, and the general quadric equation for a sphere. An alternative method for sphere intersection tests involves using the specific geometric properties of a sphere rather than solving a generic quadric equation. Designing optimised intersection algorithms for each primitive yields a valuable performance increase, but only changes overall runtime by a constant factor. Optimised algorithms for intersecting several geometric primitives are given in the Graphics Gems series [Glas90, Glas91, Glas92, Heck94, Paet95]. In order to increase performance beyond the constant factor afforded by primitive specific optimisations, techniques for reducing the number of intersection tests must be employed.

4.2.2 Bounding volumes

An important savings can be done when a complex object is bounded by a bounding volume. At the first instance intersection with bounding volume is checked. If such intersection is not found then exact intersection calculation for the object can be discarded. The bounding volume should fit as tightly as possible to the object. It means that the type of the bounding volume should be selected taking into account both the shape of the object and the cost of intersection with bounding volume. Whitted

[Whit80] originally used spheres as bounding volumes since they are one of the fastest shapes to test for intersections. Other shapes such as axis aligned bounding boxes [Rubi80], and an intersection of slabs may also be used. See figure 4.2. Weghorst *et al.* [Wegh84] pointed out that the effectiveness of a bounding volume is directly related to its tightness of fit, and its own cost of intersection.

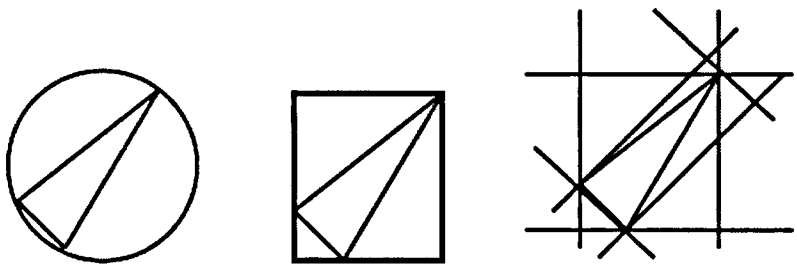


Figure 4.2: Bounding volumes (a) bounding sphere, (b) axis-aligned bounding box, (c) slabs [Watt92].

4.2.3 Bounding volumes hierarchy

A common extension to bounding volumes first suggested by Rubin and Whitted [Rubi80] and discussed in [Wegh84], is to attempt to impose a hierarchical structure on such volumes on the scene. If it is possible, objects in close spatial proximity are allowed to form clusters and the clusters are themselves enclosed in bounding volumes. This is demonstrated in figure 4.3. The ray tracer initially traverses a hierarchy of bounding volumes and a descent through this hierarchy continues only from those nodes where intersections occur. The implication here is that bounding volumes was further exploited and the intersection testing and calculation time are cut down by structuring the bounding volumes in a way that reduces the number of intersection tests. This method makes the time spent on intersection tests logarithmic rather than linear in the number of objects in the scene. In figure 4.3, ray 1 tests against three bounding volumes and one object rather than ten objects. Ray 2 tests against three bounding volumes, enters the cluster and tests against three more bounding volumes and finally three

objects. This results in a total of four object intersection tests and nine bounding volume tests compared with twenty object intersection tests for the unstructured environment.

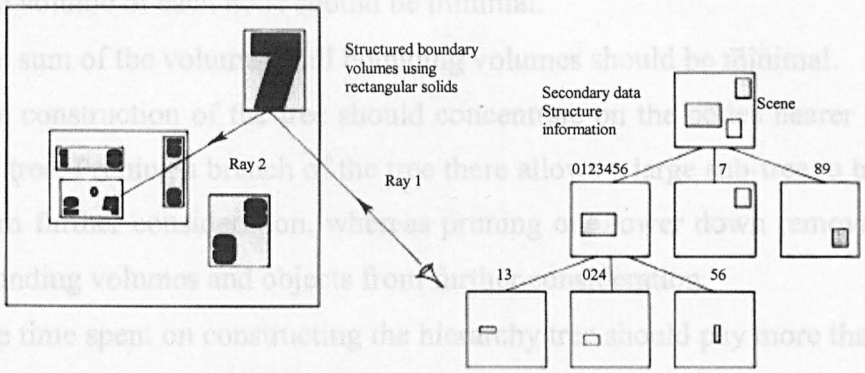


Figure 4.3: Bounding volumes hierarchy using boxes [Watt92].

Boxes have been used as bounding volumes. A box has a more difficult intersection test than a sphere. However, it is easier to set up a hierarchy with boxes than it is with spheres. Boxes can be nested inside each other using simple comparisons on each face. Thus a scene is grouped into object clusters and each of those clusters may contain other groups of objects that are spatially clustered. Ideally, high-level clusters are enclosed in bounding volumes that contain lower level clusters and bounding volumes. Clusters can only be created if objects are sufficiently close to each other. Creating clusters of widely separated objects contradicts the localisation of the process. The potential clustering and the depth of the hierarchy depend on the nature of the scene. The deeper the hierarchy is, the greater the potential saving is. Although this technique removes the linear dependence of intersection calculation overheads on scene complexity, it still has a high cost. This is because every ray descends through a tree hierarchy from the root and the bounding volumes may themselves be inefficient. Also considerable user investment is required to set up a suitable hierarchy. Kay and Kajiya [Kay86] give a list of desirable properties for any hierarchical scheme. These are:

(1) Any given sub-tree should contain objects that are near each other. "Nearness" is relative, the lower the sub-tree is with respect to the entire hierarchy, the nearer the objects should be to each other.
(2) The volume of each node should be minimal.
(3) The sum of the volume of all bounding volumes should be minimal.
(4) The construction of the tree should concentrate on the nodes nearer the root of the tree. Pruning a branch of the tree there allows a large sub-tree to be removed from further consideration, whereas pruning one lower down removes just few bounding volumes and objects from further consideration.
(5) The time spent on constructing the hierarchy tree should pay more than itself for the time saved rendering the image.

4.2.4 Space subdivision and spatial coherence

The idea behind spatial coherence schemes is simple. The space occupied by the scene is subdivided into regions, rather than a ray is checked against all objects or sets of bounded objects, it is checked against the subset of objects inside the region it is currently traveling. This group of objects is then tested for intersection with the ray. The size of the subset and the accuracy to which the spatial occupancy of the objects is determined varies, depending on the nature and number of objects and the method used for subdividing the space. Three dimensional subdivision schemes divide the space into labeled regions that are non-overlapping. If the regions are processed in order along the ray from its origin then the first object encountered is the first hit. Space subdivision can be represented in several algorithms, some of them are following:

(1) Uniform grid: It involves dividing all of the occupied space into equally sized cubic regions regardless of occupancy by objects. The cubic regions are called *voxels*. The three dimensional grid obtained in this way generates many voxels. It enables very fast tracking of rays from region to region. On the other hand, subdivision does not depend on the object structure, and thus can be very inefficient for scenes exhibiting non-uniform object distribution. In figure 4.4

ray 1 tests object 7. Ray 2 tests object 5, object 6 then objects 3, 4 and 0. The process terminates at object 4.

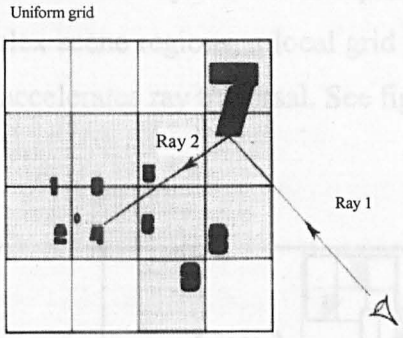


Figure 4.4: Uniform grid [Watt92].

(2) Octrees: is a hierarchical data structure that specifies the occupancy of voxels, which vary in size. The variation in size means that large empty regions, or large regions that contain a single object, are not subdivided to the same extent as regions that contain a larger number of objects. Distribution of the objects in the scene can be highly non-uniform and octree subdivision still behaves well. Its drawback is that the cost of ray traversal in octree grid is high. In figure 4.5 the space has been subdivided until each sub-region contains only a single object. Ray 1 tests object 7. Ray 2 tests object 5, object 6, and object 4. The process terminates when the first intersection is found.

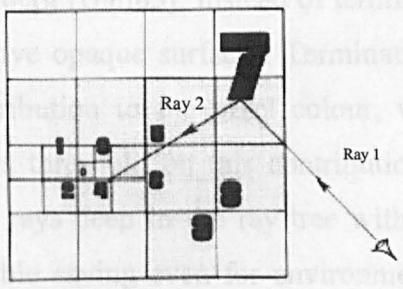


Figure 4.5: Octree subdivision [Watt92].

(3) Hierarchical grid: is a composition of uniform grids and octrees. It uses the efficient traversing of grids and the adaptation of hierarchical octrees. The adaptive grid structure enables rays to travel rapidly through empty areas. If a ray enters more complex scene regions, a local grid bounding an object or group of objects efficiently accelerates ray traversal. See figure 4.6.

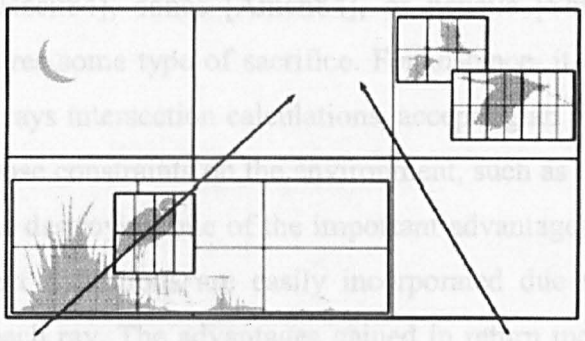


Figure 4.6: Hierarchical grid [Watt92].

4.3 Fewer rays

This category consists of techniques, which reduce the number of rays that need to be intersected with the environment. This includes primary rays as well as those created by reflection, refraction, and shadowing. The technique called *adaptive tree depth control* was introduced by Hall and Greenberg [Hall83]. Instead of terminating the ray tree at a predefined depth or at non-reflective opaque surfaces. Termination criterion took into consideration the maximum contribution to the pixel colour, which could result by continuing the recursion. Setting a threshold on this contribution made it possible to eliminate the processing of many rays deep in the ray tree without altering the result perceptibly. This led to considerable saving even for environments with many highly reflective surfaces. Other techniques that involve optimisations for antialiasing are explained in chapter 3 such as adaptive antialiasing.

4.4 Generalised rays

The difficulty of antialiasing and exploiting coherence in ray tracing comes from its use of infinitesimally thin rays. Though the simple form of these rays leads to easy representation, efficient intersection calculations, and great generality. Generalised rays traded some of these benefits in exchange for speed. The way to do this is to discard individual rays and, instead, operate simultaneously on entire families of rays, which are bundled as beams [Heck84], cones [Aman84], or pencils [Shin87]. Each of these generalised rays requires some type of sacrifice. For instance, it may need to abandon the notion of "exact" rays intersection calculations, accepting an approximation instead, or it may need to impose constraints on the environment, such as restricting the types of primitive objects, thus destroying one of the important advantages of ray tracing which is that different object definitions are easily incorporated due the separation of the intersection test for each ray. The advantages gained in return include faster execution and effective antialiasing.

4.4.1 Cone tracing

Amanatides [Aman84] generalised rays to the right circular cones which are represented by apex, centre line and spread angle. For the purpose of antialiasing, the intersection calculation not only needs to detect when a cone and an object intersect, but how much of the cone is blocked by the object. A sorted list of the closest few objects, which intersect the cone, is required so that the partial coverage can be properly combined. See figure 4.6. For reflection and refraction, the new centre line is computed using standard ray tracing techniques. See figure 4.7. The calculation of the new virtual origin and spread angle requires knowledge of the surface curvature. The method of cone tracing also extends the ray tracing to include soft shadows and dull reflections. Due to the difficulty of the cone intersection and partial coverage calculations for most objects, the environment is restricted to spheres, planes, and polygons.

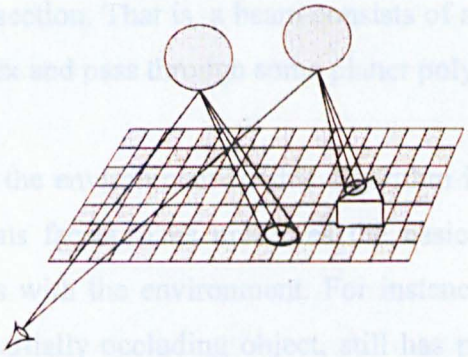


Figure 4.6: Partial coverage of objects is combined for cone tracing [MPI].

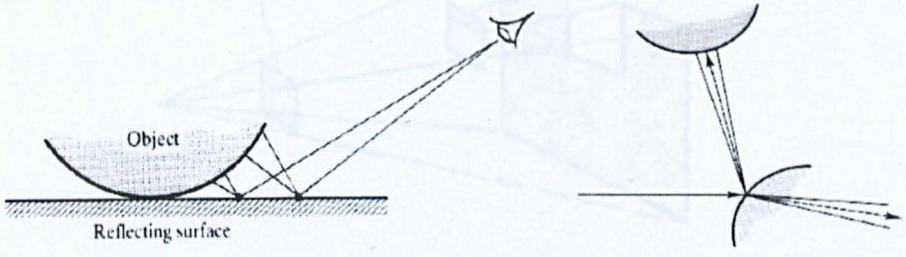


Figure 4.7: Reflection and refraction in cone tracing [MPI].

Kirk [Kirk87] extended the cone technique by accelerating the processing of partial intersection. The projected area of cone-sphere and cone-plane intersections can be pre-calculated for a wide range of cases and stored in a table. Using a look-up table instead of direct calculation produces an approximation but fast partial coverage calculation. The cone area at the intersection can also be used to properly antialias procedural textures. The cone radius at the intersection determines the aperture size of the smallest feature, which should be represented in the texture.

4.4.2 Beam tracing

Heckbert and Hanrahan [Heck84] introduced a different ray generalisation in beam tracing algorithm. In this approach rays are replaced by beams, which are cones with

arbitrary polygonal cross section. That is, a beam consists of a collection of rays, which originate at a common apex and pass through some planer polygon.

The restriction placed on the environment by this algorithm is that all objects must be constructed using polygons facets. This preserves the basic characteristics of beams under various interactions with the environment. For instance, the portion of a beam, which continues past a partially occluding object, still has polygonal cross section as shown in figure 4.8, as do beams that reflected from any surface. See figure 4.9.

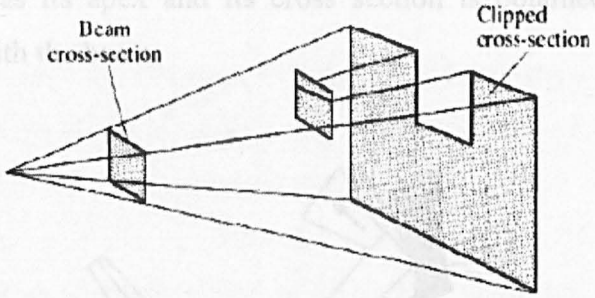


Figure 4.8: A polygon construction is clipped out of cross section of a beam [MPI].

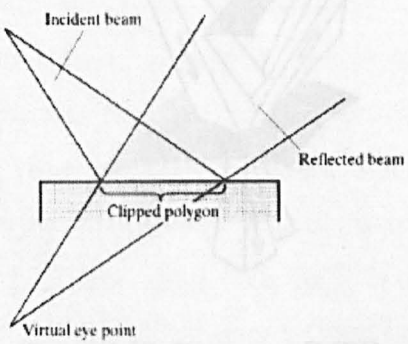


Figure 4.9: Reflected beams retain the polygonal cross section [MPI].

Refraction is the one phenomenon, which does not preserve the nature of beams. Because of non-linearity, a refracted beam can no longer be a cone. One solution is to

approximate the effect of refraction with linear transformation. This is another compromise, which must be made in order to obtain the benefits of beam tracing.

Many aspects of the beam tracing algorithms are very similar to those of standard ray tracing. In figure 4.10 a *beam tree* is constructed by recursive reflection and transmission of beams, though the process of applying these operations to beams are more complex than corresponding operations used in standard ray tree. Figure 4.9 demonstrates when reflective surfaces are encountered a "virtual eye" point is computed by reflecting the apex of the beam through the plane of the polygon. The reflected beam has the virtual eye as its apex and its cross section is obtained by intersecting the reflective polygon with the beam.

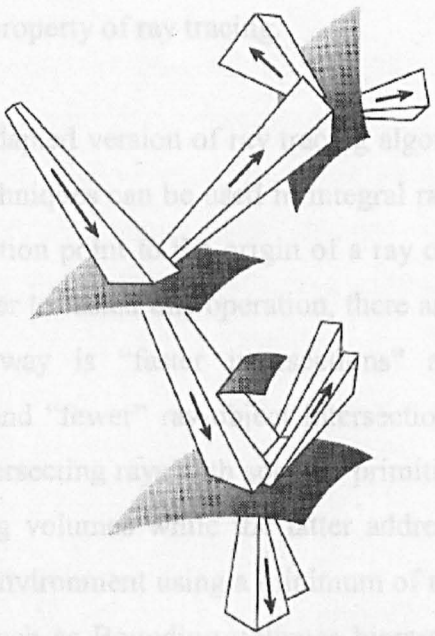


Figure 4.10: Beam tree [MPI].

4.4.3 Pencil tracing

A pencil is another type of generalised ray. It is comprised of rays, which are in the locality of a special ray called *axial ray*. Each of these nearby *paraxial* rays can be

represented as a four-dimensional vector encoding its deviation from the axial ray. By restricting attention to small deviations from the axial ray, the pencil transformation could be assumed to be linear, and therefore representable as 4 × 4 system matrices. Propagation of rays grouped as pencils could then be carried out by combining the system matrices corresponding to the individual surfaces. The approximation is only valid for sufficiently smooth surfaces, so it can only be applied to pencils, which do not encounter edges or surface discontinuities. Shinya *et al.* [Shin87] traced individual rays in the areas, which pose these problems.

4.5 Summary

In this chapter traditional acceleration techniques for ray tracing have been studied mentioning how efficiency has been the focus of the research of ray tracing due to the exhaustive computational property of ray tracing.

Integral ray tracing is an adapted version of ray tracing algorithm. Therefore traditional ray tracing acceleration techniques can be used in integral ray tracing. The operation of finding the closest intersection point to the origin of a ray consumes most of the spent time for ray tracing. In order to fasten this operation, there are three main ways that can be exploited. The first way is "faster intersections" and it separates into the subcategories of "faster" and "fewer" ray-object intersections. The former consists of efficient algorithms for intersecting rays with specific primitive objects such as efficient intersection tests, bounding volumes while the latter addresses the larger problem of intersecting a ray with an environment using a minimum of ray-object intersection tests. It consists of algorithms such as Bounding volumes hierarchy, Space subdivision and spatial coherence. Space subdivision algorithms subdivided the space occupied by the scene into regions, rather than a ray is checked against all objects or sets of bounded objects, it is checked against the subset of objects inside the region it is currently traveling. The regions are processed in order along the ray from its origin then the first object encountered is the first hit. Space subdivision can be represented in several algorithms such as uniform grid, octree and hierarchical grid.

The second way is "fewer rays", which consists of techniques that focus on reducing the number of rays to be intersected with the environment. This includes primary rays as well as those created by reflection, refraction, and shadowing. Such techniques include adaptive tree depth control and optimisation for antialiasing.

The third way of reducing the number of intersection test is called "generalised rays". Generalised rays traded some of ray tracing benefits in exchange for speed. Individual rays are discarded and replaced by entire families of rays, which are bundled as beams, cones, or pencils. Generalised rays require some type of sacrifice. For instance, they need to abandon the notion of "exact" rays intersection calculations, accepting an approximation instead. They also impose constraints on the environment, such as restricting the types of primitive objects.

In the next chapters novel ways for reducing the number of intersection tests and the computations in integral ray tracing are discussed. They use the unique approach of utilizing the properties of the integral imaging camera system in order to time-improve the exhaustive property of ray tracing. Next chapter explains how pixels can be coherently traced in order to improve shadow cache algorithm in integral ray tracing.

Chapter 5

Novel Pixel-tracing styles for Faster Shadow Cache in Integral Ray Tracing

Statistically, more intersection tests are performed for shadow determination than for primary rays, reflection rays, and transmission rays. The number of shadow rays spawned is proportional to the number of light sources. In a scene containing three light sources, approximately 75% of the rays spawned are attributable to shadow tests. In this example, 25% of the rays cast are primary rays. For each primary ray that intersects an object, surface shading calculations consider the visibility of each one of the three light sources. Shadow cache algorithm described in the next section accelerates ray tracing by reducing the number of intersection tests performed for shadow rays. Due to the unique characteristics of the camera model in integral ray tracing, the order in which pixels are traced needs to be exploited to utilise shadow cache algorithm for integral ray tracing. In this thesis the order in which pixel are traced is termed pixel-tracing. In this chapter an analysis of the relationship between rays and shadows is established in order to find out the optimal pixel-tracing style for integral ray tracing that leads to the fastest shadow intersection tests using shadow cache. Novel pixel-tracing styles are developed in order to utilise shadow cache algorithm for integral ray tracing acceleration. Pixel-tracing styles presented are compared and combined in order to produce the optimal pixel-tracing style. Experiments show resulted time improvement is up to 41% for lenticular sheets and 18.6% for micro-lens arrays. The chapter also describes how the

proposed pixel-tracing styles do not affect the scalability of running integral ray tracing on parallel computers.

5.1 Shadow caching

The shadow cache algorithm was introduced by Hains [Hain86], where image-space coherence is used to decrease the number of the shadowing intersection tests. They considered the observation that if an object prevents a light source from contribution to the intensity of a pixel, then the same object is likely to block the same light for an adjacent pixel. Shadow cache also operates on the assumption that consecutive rays share some amount of spatial coherence and that statically they tend to intersect objects at points which are near previous intersections.

Shadow cache works by keeping a cache of the most recent shadow casting object for each light source. The next time a light is tested for occlusion, the associated shadow casting object is first tested for intersection rather than traversing the objects within the scene in the usual manner as illustrated in figure 5.1. If the shadow test of the cached object fails, then testing must continue on the objects in the usual manner. If the shadow test on the cached object succeeds, significant computation can be saved.

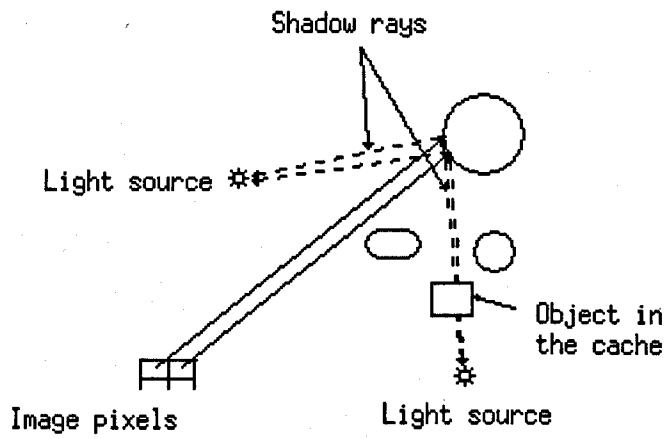


Figure 5.1: Shadow cache.

Computer generation of integral imaging requires higher resolution display i.e. larger number of pixels than normal computer images in order to produce an integral image with acceptable quality [Min01]. The efficiency of the shadow cache increases as the resolution of the image increases [Hain86b] The reason for this is when an environment is rendered at higher resolution, shadow casting objects prevents light sources from contribution to the intensity of a larger number of pixels. In order to apply shadow cache to integral ray tracing, it requires coherent pixel-tracing style that ensures successively tracing of pixels whose primary rays share spatial coherence [Osam02]. That increases the number of shadow rays that utilise the shadow cache and saves large number of intersection tests.

5.2 Cache calculations

While using the shadow cache algorithm, if a shadow ray is tested against the object in the cache and the test succeed, this condition is termed as cache hit. If a shadow ray does not achieve a cache hit, the number of intersection tests for this shadow ray can be in thousands or in millions depending on the scene complexity while a shadow ray achieves a cache hit, is tested only once for intersection with the object in the cache.

Let N be the number of shadow rays whose pixels are in shadow, C_m be the average computation cost of the intersection tests required when a shadow ray does not achieve a cache hit, and C_h be the average computation cost for the intersection test required when a shadow ray achieves a cache hit. The total computation cost for shadow rays C_t can be expressed as follow:

$$C_t = N_h C_h + N_m C_m, \tag{5.1}$$

where N_h is the number of shadow rays that achieve a cache hit and N_m is the number of shadow rays that do not achieve a cache hit. The total number of the shadow rays N can be expressed as follows:

$$N = N_h + N_m.$$

Since, $C_m >> C_h$, the total computation cost C_t can be written as follows:

$$C_t = N_m C_m \tag{5.2}$$

Equation 5.2 shows that in order to minimize the total computation cost for shadow rays, the number of shadow rays achieving cache hits must increase.

Consider a number of pixels N whose primary rays intersect shadowed surfaces. Considering one light source, the number of shadow rays intersecting with a shadow casting object is M_h and their associated pixels are termed as H pixels. The number of shadow rays that do not intersect with that shadow casting object is M_m and their associated pixels are termed as M pixels. M and H pixels belong to shadowed areas, therefore:

$$N = M_h + M_m \tag{5.3}$$

The pixel-tracing style has a great effect on C_t. Obviously the best case is when H pixels are traced successively before switching to M pixels. This results that the shadow casting object is being kept in the cache once it is intersected and the only pixels which do not achieve a cache hit are M pixels. In this case:

$$N_m = M_m, \text{ and } N_h = N - M_h$$

Substituting in equation 5.2 gives:

$$C_t = M_m C_m \tag{5.4}$$

The worst case is when pixels of each group are traced in alternating order i.e. H pixel then M pixel then H pixel and so on, which results in a different object is being stored in the cache each time a pixel is traced. In this case for each M pixel, two shadow rays do not achieve a cache hit. Therefore:

$$N_m = 2M_m,$$

which is double the number of shadow rays of the best case that do not achieve a cache hit, and therefore N_h can be written as follows:

$$N_h = N - 2M_h$$

Substituting in equation 5.2 gives:

$$C_t = 2M_m C_m \tag{5.5}$$

Equation 5.5 shows that tracing pixels in a non-coherent way is very computationally expensive and could duplicate the computation cost of shadow rays.

The next section describes the implication of the integral camera model on the shadow cache algorithm and the spatial coherence between primary rays.

5.3 Integral camera model implications

Due to the nature of the recording process of integral imaging, many changes on the camera model in ray tracing are required. For lenticular sheets, each lens acts like a cylindrical camera. A strip of pixels is associated with each lens forming a sub-image. Each lens records a sub-image of the scene from a different angle as shown in figure 5.2. For micro-lens arrays each lens acts like a square or a hexagonal camera depends on the structure of the lenses as shown in figure 5.3. In the lateral cross section of the lenticular or the micro-lenses, the model appears as a pinhole. In the case of lenticular sheets, the pinhole forms a straight line parallel to the axis of the cylindrical lens in the vertical direction that makes an axial camera model. For each pixel, a primary ray is spawned. The recording path of the primary ray draws a straight line going forward towards the image plane and backward away from the image plane. Similar primary rays of neighbour lenses are spawned to similar directions parallel to each other. Therefore correlated sub-images are produced which is a property of an integral imaging.

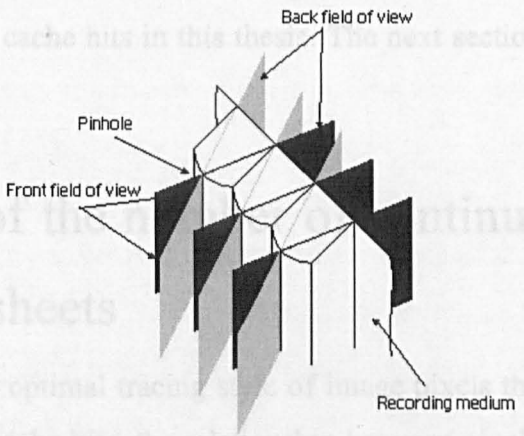


Figure 5.2: Lenticular sheet model in integral ray tracer.

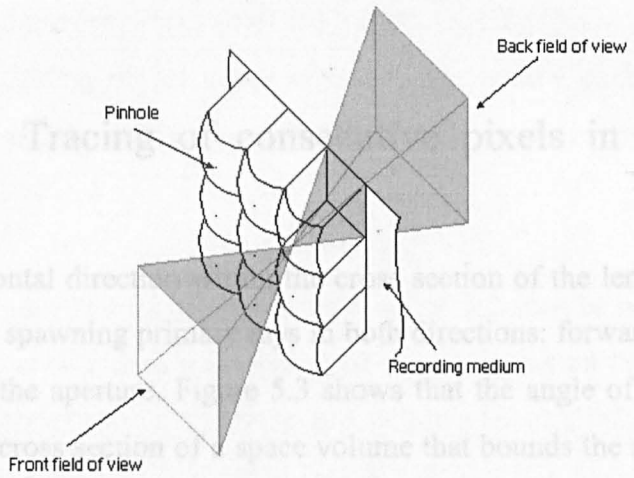


Figure 5.3: Micro-lens array in integral ray tracing.

The structure of the lenses and the camera model in the integral ray tracing affects the way primary rays are spawned as well as the spatial coherence among them. From the shadow cache calculations, it is shown that tracing all of H pixels before switching to M pixels maximises the number of cache hits and accordingly minimises the computation cost. In the next two sections an analysis giving the relationship between rays and shadowed surfaces is carried out for different pixel-tracing styles in order to find the optimal pixel-tracing style considering the number of H pixels traced consecutively as a measure of the efficiency. The number of H pixels traced consecutively is represented by the term continuous cache hits in this thesis. The next section describes the analysis for lenticular sheets.

5.4 Analysis of the number of continuous cache hits for lenticular sheets

In order to find out the optimal tracing style of image pixels that attains the maximum number of continuous cache hits, the relationship between pixels and their primary rays and shadows is analysed. The analysis consists of the way primary rays relate to the shadow size and location and its affect on the number of continuous cache hits. For

simplification, it is assumed that the shadow is projected on a flat surface facing the camera model.

5.4.1 Style 1: Tracing of consecutive pixels in the horizontal direction

Pixels in the horizontal direction within the cross section of the lenses are traced from the lens pinhole by spawning primary rays in both directions: forward to the pixels, and backward towards the aperture. Figure 5.3 shows that the angle of view of the lens α forms a horizontal cross section of a space volume that bounds the rays traced from the lens through the object space. α is defined as:

$$\alpha = 2 \times tan^{-1} (P/2f) \tag{5.6}$$

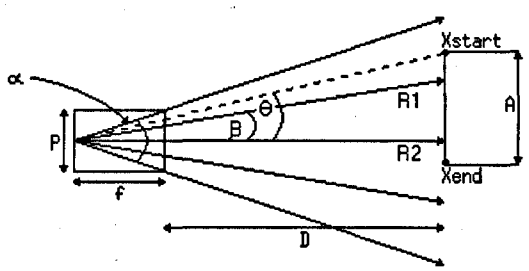


Figure 5.3: lateral cross section of a lens in integral ray tracing.

where P and f are the pitch and the focal length of the lens respectively. The number of pixels behind each lens is N_{lens} which is defined as:

$$N_{lens} = P/p_s$$

where p_s is the pixel size. In the case of uniform sampling of the space, pixel size in the horizontal direction is the same for the vertical one therefore the spacing between the centres of the adjacent pixels equals the pixel size p_s. Assuming that the angle between two adjacent rays β is small enough (for a typical lenticular sheet $\beta = 0.037$ Rad), β can be defined as:

$$\beta = \alpha/N_{lens} \tag{5.7}$$

Figure 5.3 shows a shadow area of a width A is projected on some surface at distance D from a lens and a primary ray $R1$ is spawned intersecting the shadow area at some point, causing a shadow casting object to be saved in the shadow cache. The angle θ is between two rays, which covers the width A of the shadow casting object. θ can be written as:

$$\theta = tan^{-1} A/D \tag{5.8}$$

In order that the primary ray $R2$ intersects the shadow, $R2$ must be bounded by the angle θ (i.e. $\beta < \theta$). If $\theta < \alpha$ then the number of rays intersecting that shadow is $\lfloor \theta/\beta \rfloor$. Therefore, the number of continuous cache hits within a lens $CCWL_{hit}$ can be written as:

$$CCWL_{hit} = \lfloor \theta / \beta \rfloor \tag{5.9}$$

Substituting equation 5.6 and equation 5.8 in equation 5.9 yields:

$$CCWL_{hit} = N_{lens} \times (tan^{-1} (A/D)) / (2 \times tan^{-1}(N_{lens} \times p_s/2f)) \tag{5.10}$$

Therefore $CCWL_{hit}$ is a function of A and D.

While tracing pixels in the horizontal direction across the lenses, switching between lenses has a big effect on the shadow cache status whether the shadow ray being processed achieves a cache hit or not. The shadow cache status depends on whether the first ray of the adjacent lens attains a cache hit or not.

In figure 5.4, $R1$ is the primary ray passing through the last pixel covered by a lens, which intersects a shadowed surface with width A that starts and ends at positions X_{start} and X_{end} respectively along the x axis. $R2$ is the primary ray passing through the first pixel covered by the adjacent lens. The position of the intersection point of $R2$ and the shadowed along the x axis X_2 can be written as:

$$X_2 = LA_{start} + P \times n - D \times tan(\alpha/2) \tag{5.11}$$

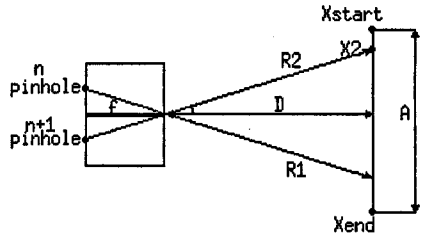


Figure 5.4: Crossing to an adjacent lens.

where LA_{start} is the starting point of the lenticular sheet along the x axis, and n is the lens index. $R2$ intersects the object stored in the cache and achieves a cache hit under the condition:

$$X_2 \geq X_{start}$$

where $X_{start} = X_{end} - A$.

By substituting equation 5.11 the condition becomes:

$$[P \times n + LA_{start} - D \times tan\,(\alpha/2)] \geq [X_{end} - A] \tag{5.12}$$

Therefore the shadow cache status for crossing from a lens to an adjacent one $CSTAL$ is a function of A and D, and has two cases as follows:

$$CSTAL = \begin{cases} 1 & X_2 \geq X_{start} \\ 0 & X_2 < X_{start} \end{cases} \tag{5.13}$$

Considering multiple lenses as shown in figure 5.5. Then the number of continuous cache hits per lens CCL_{hit} is described as:

$$CCL_{hit} = CCWL_{hit} + CSTAL \tag{5.14}$$

CCL_{hit} also is a function of A and D because $CCWL_{hit}$ and $CSTAL$ are function of A and D. Therefore, the number of continuous cache hits $CC_{hit\ style1}$ for tracing consecutive pixels in the horizontal direction is represented as:

$$CC_{hit\ style1} = CCL_{hit\ 0} + \sum_{lensID=1}^{n} CSTAL_{n-1} \times CCL_{hit\ n} \tag{5.15}$$

which is a function of A and D.

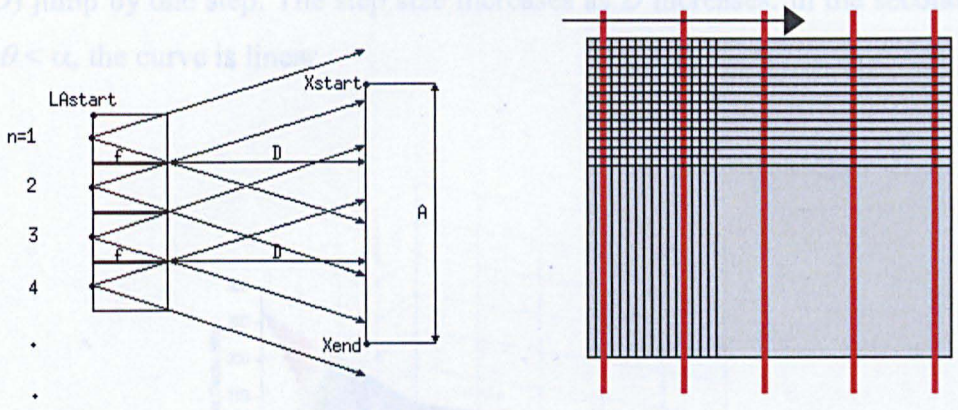


Figure 5.5: Tracing consecutive pixels in the horizontal direction.

- The number of continuous cache hits

Because $CC_{hit\ style1}$ is a function of A and D, it is represented graphically by a surface as shown in figure 5.6. For each value of D there is a function $CC_{hit}(A)$ represented by a curve on that surface. The value of $CC_{hit}(A)$ increases when D decreases. $CC_{hit}(A)$ reaches its maximum when $D = 0$. The curve $CC_{hit}(A)_{D=0}$ represents a linear function and equals to A/ps, which is the same as CC_{hit} obtained for style 3. When D is not 0, $CC_{hit}(A)$ curve consists of two regions as shown in figure 5.7. In the first region, the curve is linear because not every ray in the first lens intersects points in shadow ($\alpha < \theta$). In this region, $CC_{hit}(A)$ is less than $(N_{lens} - 1)$.) The second region exists when $\theta \geq \alpha$ in this region, the curve is similar to a staircase that increases as A increases. The step size depends on D. The reason for the existence of the steps is that when the last ray in a lens intersects or misses points in shadow. The number of rays of the adjacent lens intersecting points in shadow is added to or subtracted from CC_{hit}, which makes $CC_{hit}(A)$ jump by one step.

Also for each value of A there is a function $CC_{hit}(D)$ represented by a curve as shown in figure 5.8. The value of $CC_{hit}(D)$ decreases as D increases. This curve also consists of two regions but in an opposite way to $CC_{hit}(A)$. The first section is a staircase decreasing as D increases. When D is varied by $2f$, A covers the last ray in a lens, which makes

$CC_{hit}(D)$ jump by one step. The step size increases as D increases. In the second region where $\theta < \alpha$, the curve is linear.

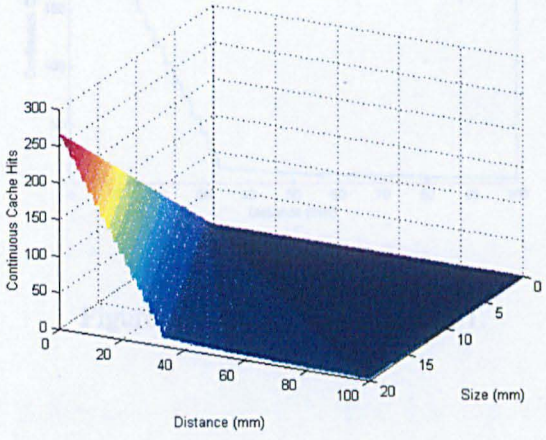


Figure 5.6: $CC_{hit}(A,D)$ of style 1.

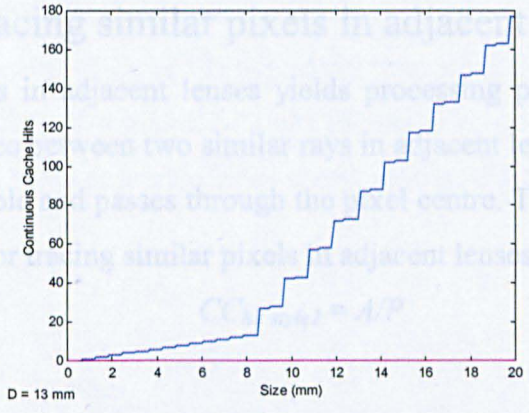


Figure 5.7: $CC_{hit}(A)_{D=13}$ of style 1.

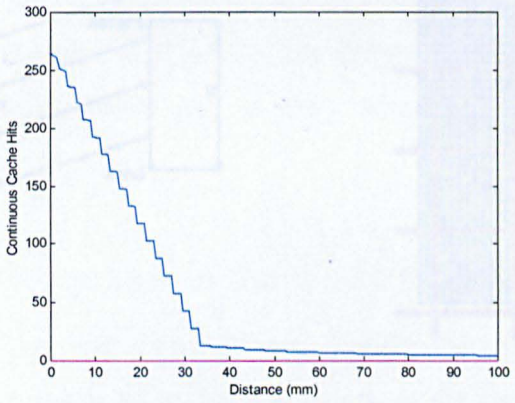


Figure 5.8: $CC_{hit}(D)_{A=20}$ of style 1.

Style 1 has some limitations in terms of continuous cache hits, which is a result of the discontinuity that happens at the edge of each lens as well as not considering the second dimension while tracing pixels.

5.4.2 Style 2: Tracing similar pixels in adjacent lenses

Tracing similar pixels in adjacent lenses yields processing parallel rays as shown in figure 5.9. The distance between two similar rays in adjacent lenses is P, where each ray starts at the lens pinhole and passes through the pixel centre. The number of continuous cache hits $CC_{hit\ style2}$ for tracing similar pixels in adjacent lenses can be written as:

$$CC_{hit\ style2} = A/P \tag{5.16}$$

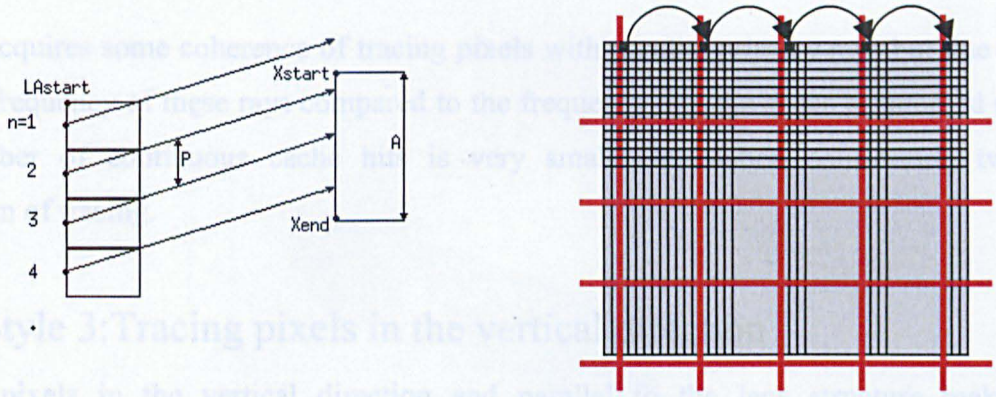


Figure 5.9: Tracing similar pixels in adjacent lenses

Therefore $CC_{hit\ style2}$ is a function of A only. Notice that P/p_s could be 10, 20 or bigger depending on the image resolution. In this style the traced rays sample the scene at very low sampling rate and thus it acquires a small number of continuous cache hits even considering two dimension of tracing.

- The number of continuous cache hits

$CC_{hit\ style2}$ is not a function of D, and it is linearly proportional to A because traced rays are parallel to each other. Therefore $CC_{hit}(A)$ increases or decreases, if and only if, A increases or decreases by an amount of P as shown in figure 5.10.

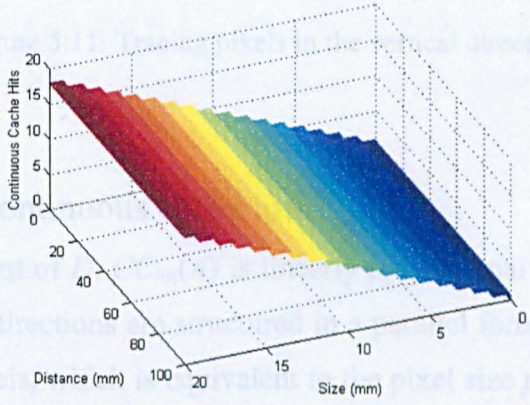


Figure 5.10: $CC_{hit}(A)$ of style 2.

Style 2 acquires some coherence of tracing pixels with parallel primary rays but due to the low frequency of these rays compared to the frequency that the scene is sampled at, the number of continuous cache hits is very small even when considering two dimension of tracing.

5.4.3 Style 3:Tracing pixels in the vertical direction

Tracing pixels in the vertical direction and parallel to the lens structure makes successive rays parallel to each other. Each ray starts at the centre of the pixel and goes in the direction normal to the image plane as shown in figure 5.11. The number of continuous cache hits $CC_{hit\ style3}$ for this style can be written as:

$$CC_{hit\ style3} = A/p_s \tag{5.17}$$

Therefore $CC_{hit\ style3}$ is a function of A only.

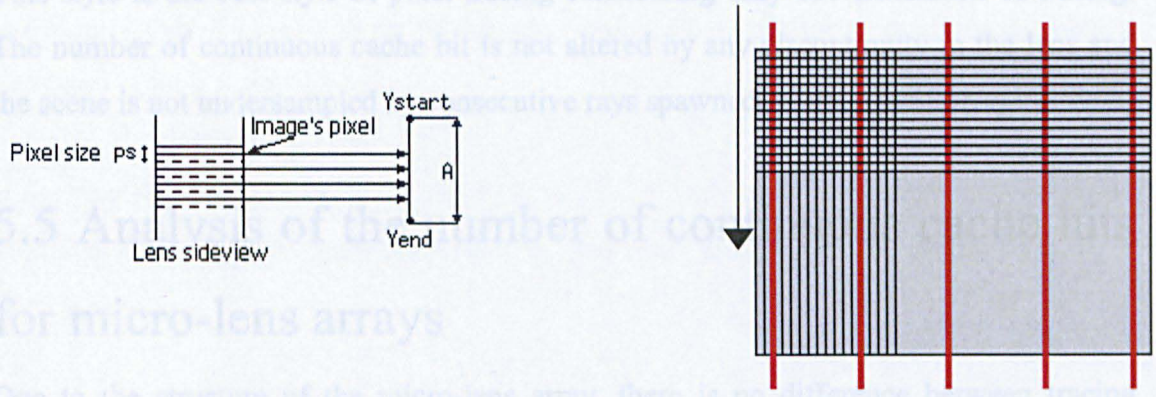


Figure 5.11: Tracing pixels in the vertical direction.

- The number of continuous cache hits

$CC_{hit\ style3}$ is independent of D. $CC_{hit}(A)$ is linearly proportional to A as shown in figure 5.12. Rays in vertical directions are structured in a parallel form spaced by the distance between centres of pixels, which is equivalent to the pixel size ps. As A varies, $CC_{hit}(A)$ is altered by adding or subtracting the number of new rays that intersect or miss points in shadow.

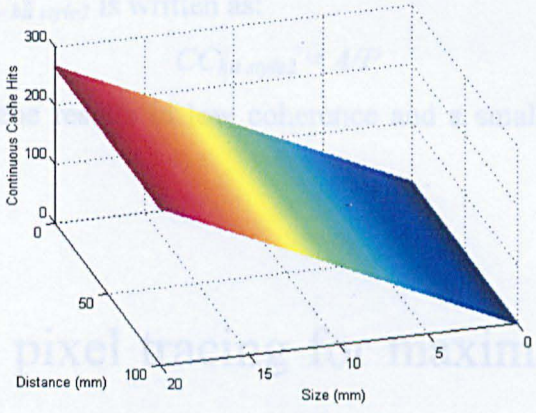


Figure 5.12: $CC_{hit}(A)$ of style 3.

This style is the best style of pixel-tracing considering only one dimension of tracing. The number of continuous cache hit is not altered by any discontinuity in the lens and the scene is not undersampled by consecutive rays spawned.

5.5 Analysis of the number of continuous cache hits for micro-lens arrays

Due to the structure of the micro-lens array, there is no difference between tracing pixels in the horizontal direction and tracing pixels in the vertical direction. Therefore style 1 and style 2 is applied in both the horizontal and the vertical directions.

5.5.1 Style 1: Tracing of consecutive pixels

As discussed in the previous section and from equation 5.15 $CC_{hit\ style1}$ is defined as:

$$CC_{hit\ style1} = CCL_{hit\ 0} + \sum_{lensID=1}^{n} CSTAL_{n-1} \times CCL_{hit\ n}$$

In style 1, the discontinuity that happens at the edge of each lens affects the coherence as well as not considering the second dimension while tracing pixels.

5.5.2 Style 2: Tracing similar pixels in adjacent lenses

From equation 5.16 $CC_{hit\ style2}$ is written as:

$$CC_{hit\ style2} = A/P$$

Undersampling the scene results in low coherence and a small number of continuous cache hits.

5.6 Optimum pixel tracing for maximum number of continuous cache hits

5.6.1 Missing a cache hit while tracing pixels

Missing a cache hit even once during tracing pixels can be very expensive as described earlier. If there are a number of pixels whose rays intersect points in shadow, the optimum style of pixel-tracing is the one, which traces all those pixels successively before switching to other pixels.

Assuming the case shown in figure 5.13. N_{lens} = 5 and there are six rays that intersect points in shadow, three from the first lens, two from the second lens and one from the third lens.

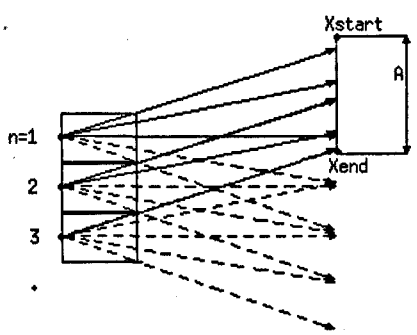


Figure 5.13: Tracing pixels.

I) Style 1:

Using style 1 yields $CC_{hit\ style1}$ being equal to three out of six possible cache hits as shown in figure 5.14. It misses cache hits before tracing all pixels whose rays intersect points in the same shadow

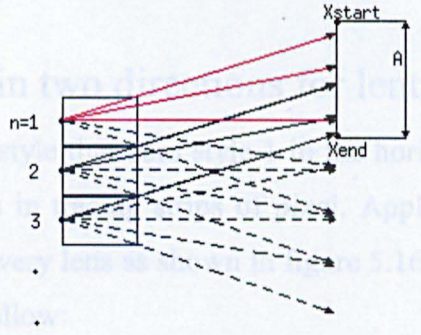


Figure 5.14: Style 1 of tracing pixels.

II) Style 2:

In style 2, $CC_{hit\ style2}$ is equal to three as shown in figure 5.15.

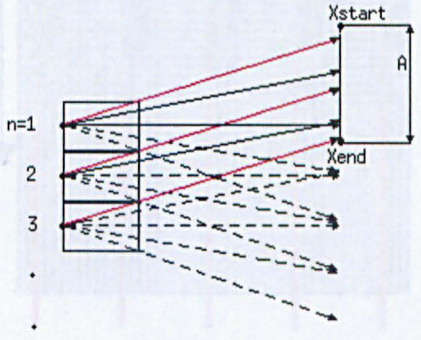


Figure 5.15: Style 2 of tracing pixels.

III) Style 3:

Tracing in the vertical direction is not affected by the properties of the lenses. Therefore, the number of continuous cache hits is found to be always equal to the number of rays that intersect points in the same shadow, which is six (the maximum achievable for one dimension).

5.6.2 Tracing pixels in two directions for lenticular sheet

Having a two-dimensional style that uses style 1 in the horizontal direction and style 3 in the vertical style, results in tracing strips of pixel. Applying the concept of style 2 leads to tracing a strip for every lens as shown in figure 5.16. The number of continuous cache hits is expressed as follow:

$$CC_{hit} = \sum_{\text{lensID}=1}^{n} CCL_{hit\ n} \times CC_{hit\ style3\ n} \tag{5.18}$$

In figure 5.17, $CC_{hit}(A,D)$ of tracing strips of pixels does not encounter any discontinuity or staircase effect.

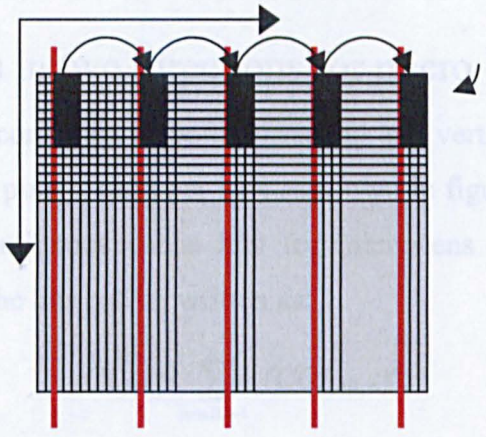


Figure 5.16: Tracing strips of pixels.

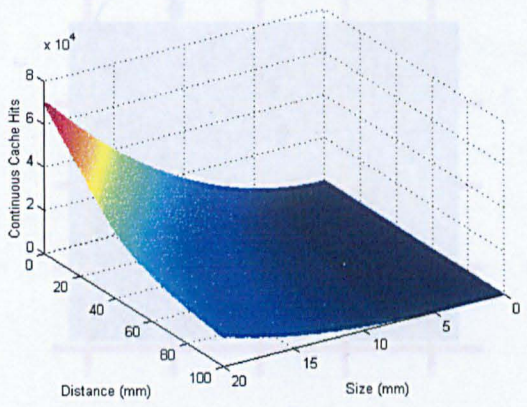


Figure 5.17: $CC_{hit}(A,D)$ of tracing strips of pixels

Assume there is a shadow projected as a rectangle, which is intersected by six rays in the horizontal direction and six rays in the vertical direction. This gives a total of thirty-six rays. Using tracing of the strips of pixels results in CC_{hit} being equal to thirty-six. Theoretically, this gives the optimum pixel-tracing style.

5.6.3 Tracing pixels in two directions for micro-lens array

If style 1 and style 2 are combined in both horizontal and vertical directions the result will be tracing squares of pixels, each per lens as shown in figure 5.18. In this case the maximum number of continuous cache hits for micro-lens array is achieved. The number of continuous cache hits can be written as:

$$CC_{hit} = \sum_{\text{lensID}=1}^{n} (CCL_{hit\ n})^2 \tag{5.19}$$

Figure 5.19 shows $CC_{hit}(A,D)$ for tracing squares of pixels with no discontinuity or staircase effect.

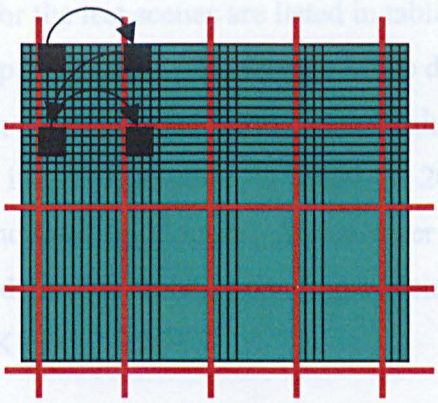


Figure 5.18: Tracing squares of pixels.

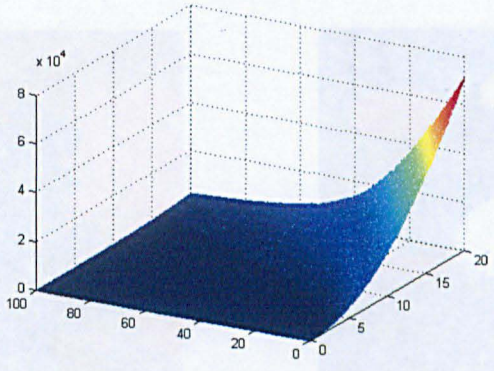


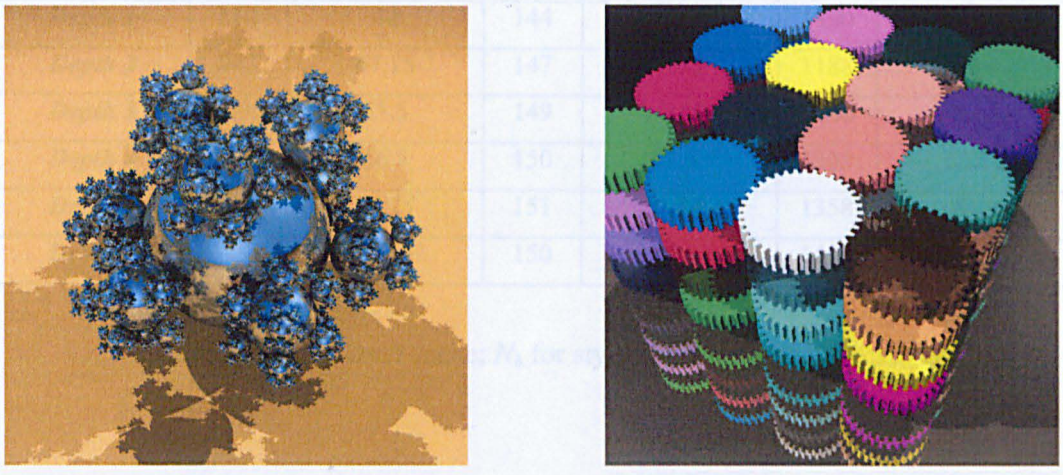
Figure 5.19: $CC_{hit}(A,D)$ of tracing squares of pixels

5.7 Tests and Results

Following is the description of the experiment settings: The lens parameters of the modelled camera system are the same as that of the lenticular sheet used in displaying the output images. The relevant parameters are: lens's pitch = 1.124 mm, lens's refractive index = 1.6, and focal lens = 3.03 mm. The same single shadow projected on a flat square facing the lens sheet, which is used in the theory is also used in testing. The different styles of pixel-tracing have been tested on benchmark scenes [Eric87] shown

in figure 5.20. The details for the test scenes are listed in table 5.1. The scenes under test are rendered at different depth distances with respect to the displaying lens sheet. These distances are 2 cm in front, 1 cm in front, 0 cm, 1 cm behind, 2 cm behind, and 3 cm behind respectively. The image resolution is 2000 × 2000 and N_{lens} = 15. All experiments have been conducted on Compaq AlphaServer DS20 with two 500 MHz Alpha 21264 processors and 1 GB memory, operating system Digital UNIX V4.0F, and the use of POSIX threads [Klei96, Nich96]

Primary tests are made on the first three styles of tracing in order to have approximated comparison between them regarding N_h before adding shadow cache algorithm to the integral ray tracing. Table 5.2 shows N_h for the three styles for each scene at different distances from the lenticular sheet. Style 3 has the greatest value for N_m.



Balls scene Gears scene

Figure 5.20: Test scenes.

Scene	No. Objects	No. Light Sources	No. Plane	No. Spheres	No. Polygons
Balls	7385	3	1	7381	
Gears	36615	6			36609

Table 5.1: Test scenes.

Depth of scene	N_h *for style 1*	N_h*% for style 1*	N_h *for style 2*	N_h*% for style 2*	N_h *for style 3*	N_h*% for style 3*
Depth 1	205	10.25	34	1.7	279	13.95
Depth 2	233	11.65	36	1.8	300	15
Depth 3	295	14.75	37	1.85	317	15.85
Depth 4	367	18.35	38	1.9	340	17
Depth 5	440	22	39	1.95	365	18.25
Depth 6	512	25.6	40	2	392	19.6

Table 5.2(a): Balls scene; N_h for style 1, style 2, and style 3.

Depth of scene	N_h *for style 1*	N_h*% for style 1*	N_h *for style 2*	N_h*% for style 2*	N_h *for style 3*	N_h*% for style 3*
Depth 1	880	44	144	7.2	1130	56.5
Depth 2	943	47.15	147	7.35	1188	59.4
Depth 3	1030	51.5	149	7.45	1246	62.3
Depth 4	1136	56.8	150	7.5	1303	65.15
Depth 5	1265	63.25	151	7.55	1358	67.9
Depth 6	1397	69.85	150	7.5	1408	70.4

Table 5.2(b): Gears scene; N_h for style 1, style 2, and style 3.

The results of testing the different styles of tracing gave identical CC_{hit}s to those in the theory as shown in figure 5.19.

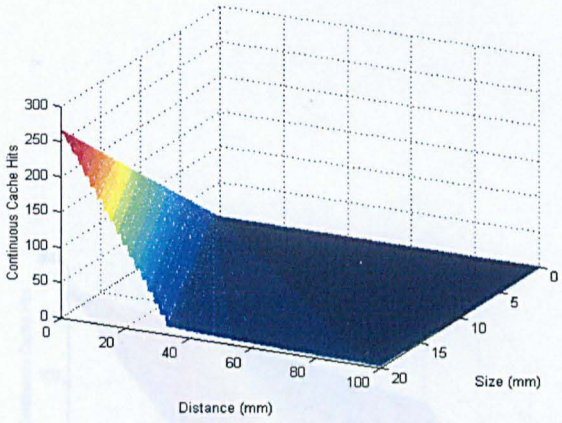


Figure 5.19(a): Continuous cache hits for style 1 for lenticular sheet.

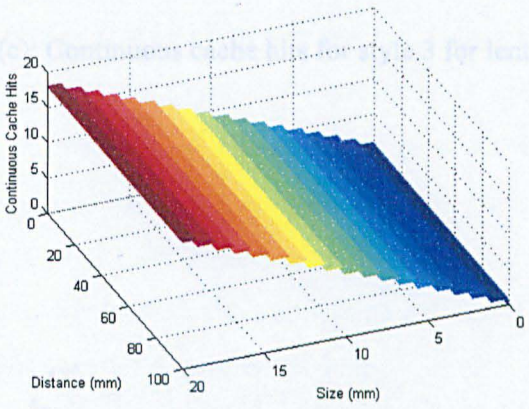


Figure 5.19(b): Continuous cache hits for style 2 for lenticular sheet.

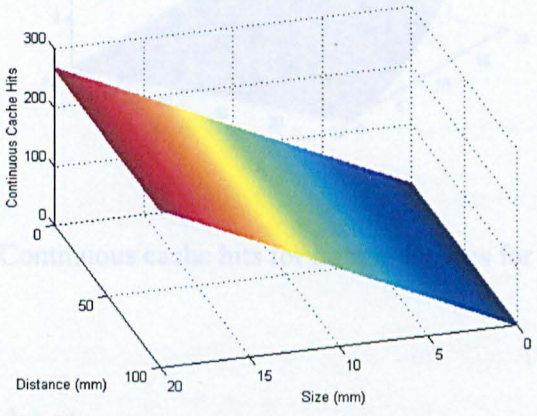


Figure 5.19(c): Continuous cache hits for style 3 for lenticular sheet.

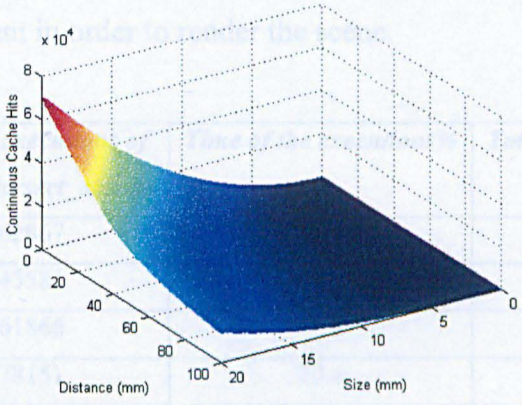


Figure 5.19(d): Continuous cache hits for tracing strips of pixels for lenticular sheet.

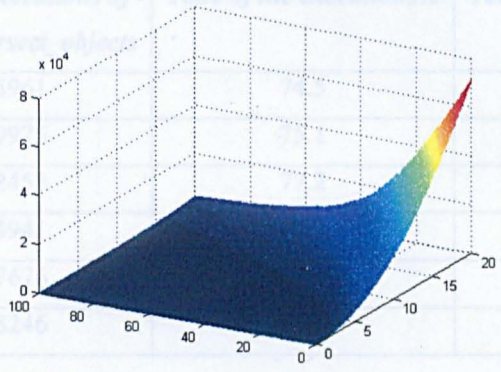


Figure 5.19(e): Continuous cache hits for tracing squares for micro-lens array.

5.7.1 Lenticular sheet

Testing on benchmark scenes showed that shadow calculations are the most expensive in terms of processing power and time. Code profiling is used in order to show the number of times *shadow_intersect_objects* function is executed, which is responsible for shadow calculations in Integral ray tracing, and the percentage of time used by *shadow_intersect_objects*. Table 5.3 shows the number of times *shadow_intersect_objects* is executed, the percentage of time it used, and the total time in seconds, which is spent in order to render the scene.

Depth	***Number of executions of shadow_intersect_objects***	***Time of the executions%***	***Total time for rendering***
1	13962867	66.9	30.0
2	14045581	67.3	31.1
3	14361866	68.8	33.2
4	14678151	70.4	35.2
5	15014044	71.9	37.8
6	15357867	73.6	41.0

Table 5.3(a): Shadow calculations for balls scene.

Depth	*Number of executions of shadow_intersect_objects*	*Time of the executions%*	*Total time for rendering*
1	22985961	74.5	124.0
2	23169975	75.1	125.0
3	23792458	77.2	135.2
4	24414941	79.2	142.0
5	25367676	82.3	148.0
6	26478246	85.9	156.0

Table 5.3(b): Shadow calculations for gears scene.

Applying shadow cache to Integral ray tracing reduces the number of executions of *shadow_intersect_objects* function. Tables 5.4(a-d) and 5.4(e-h) show the result of applying shadow cache to the balls and the gears scenes respectively. For the balls scene, the shadow cache saves up to *15.4%, 18%, 29.2%,* and *28.7%* of the executions of the function in styles 1, 2, 3, and tracing strips of pixels respectively. For gears scene, it saves up to *9.9%, 6.2%, 41.5%,* and *39.7%* of the executions of the function in styles 1, 2, 3, and tracing strips of pixels respectively. The tables also show in percentage the saved time due to applying shadow cache and the total time of rendering in seconds for each of the different styles.

Depth	*Number of executions of shadow_intersect_objects*	*Saved executions%*	*Saved time%*	*Total time for rendering*
1	11813863	15.4	5.7	28.3
2	11883435	15.4	6.4	29.1
3	12145138	15.4	7.8	30.6
4	12423066	15.4	8.5	32.2
5	12707556	15.4	9.8	34.1
6	13001778	15.3	11.9	36.1

Table 5.4(a): Style 1: saved shadow calculations for balls scene.

Depth	*Number of executions of shadow_intersect_objects*	*Saved executions%*	*Saved time%*	*Total time for rendering*
1	12006544	14	3	29.1
2	12082531	14	3.5	30
3	12168314	15.3	6	31.2
4	12283410	16.3	8	32.4
5	12425738	17.2	10.6	33.8
6	12590747	18	13.9	35.3

Table 5.4(b): Style 2: saved shadow calculations for balls scene.

Depth	*Number of executions of shadow_intersect_objects*	*Saved executions%*	*Saved time%*	*Total time for rendering*
1	10703894	23.3	13	26.1
2	10709978	23.7	12.9	27.1
3	10729748	25.3	15.7	28
4	10768080	26.6	17.3	29.1
5	10819536	27.9	18.5	30.8
6	10880696	29.2	23	31.6

Table 5.4(c): Style 3: saved shadow calculations for balls scene.

Depth	*Number of executions of shadow_intersect_objects*	*Saved executions%*	*Saved time%*	*Total time for rendering*
1	10757583	23	12.7	26.2
2	10766881	23.3	14.5	26.6
3	10790644	24.9	16.6	27.7
4	10833009	26.2	17.9	28.9
5	10889241	27.5	20.4	30.1
6	10956209	28.7	22.4	31.8

Table 5.4(d): Tracing strips of pixels: saved shadow calculations for balls scene.

Depth	*Number of executions of shadow_intersect_objects*	*Saved executions%*	*Saved time%*	*Total time for rendering*
1	20720594	9.9	3.7	119.4
2	20885155	9.9	4.1	119.9
3	21431574	9.9	4.5	129.1
4	22267890	8.8	5.3	134.5
5	23242755	8.4	6.2	138.8
6	24264986	8.4	6.3	146.2

Table 5.4(e): Style 1: saved shadow calculations for gears scene.

Depth	*Number of executions of shadow_intersect_objects*	*Saved executions%*	*Saved time%*	*Total time for rendering*
1	22377681	2.6	-9.3	135.5
2	22876224	1.3	-11.4	139.3
3	23380312	1.7	-5.6	142.8
4	23876382	2.2	-2.7	145.8
5	24370379	3.9	-4.19	154.2
6	24823603	6.2	2.2	152.5

Table 5.4(f): Style 2: saved shadow calculations for gears scene.

Depth	*Number of executions of shadow_intersect_objects*	*Saved executions%*	*Saved time%*	*Total time for rendering*
1	14657063	36.2	31.5	85
2	14858904	35.8	30.6	86.8
3	15043914	36.8	34.7	88.3
4	15204516	37.7	36.8	89.7
5	15361545	39.4	39	90.3
6	15479136	41.5	41	92.1

Table 5.4(g): Style 3: saved shadow calculations for gears scene.

Depth	*Number of executions of shadow_intersect_objects*	*Saved executions%*	*Saved time%*	*Total time for rendering*
1	15054334	34.5	29.1	87.9
2	15270643	341	28.7	89.1
3	15471888	35	33	90.6
4	15649809	35.9	36.1	90.7
5	15822953	37.6	36.1	94.5
6	15957828	39.7	38.3	96.3

Table 5.4(h): Tracing strips of pixels: saved shadow calculations for gears scene.

Figure 5.20 below shows the improvement of time in percentage for style 2, style 3, and tracing strips of pixels over style 1, at different positions of the centre of the scene with respect to the lenticular sheet. Both vertical tracing and tracing strips of pixels gives very good results over styles 1 and 2 for all scenes and reached 37% time-improvement in the gears scene. Styles 1 and 2 give similar improvements except for the gears scene where style 1 gives better results. However tracing strips of pixels does not achieve much improvement over style 3. That is because of the diverse sizes of the projected shadows and its positions with respect to the lenticular sheet.

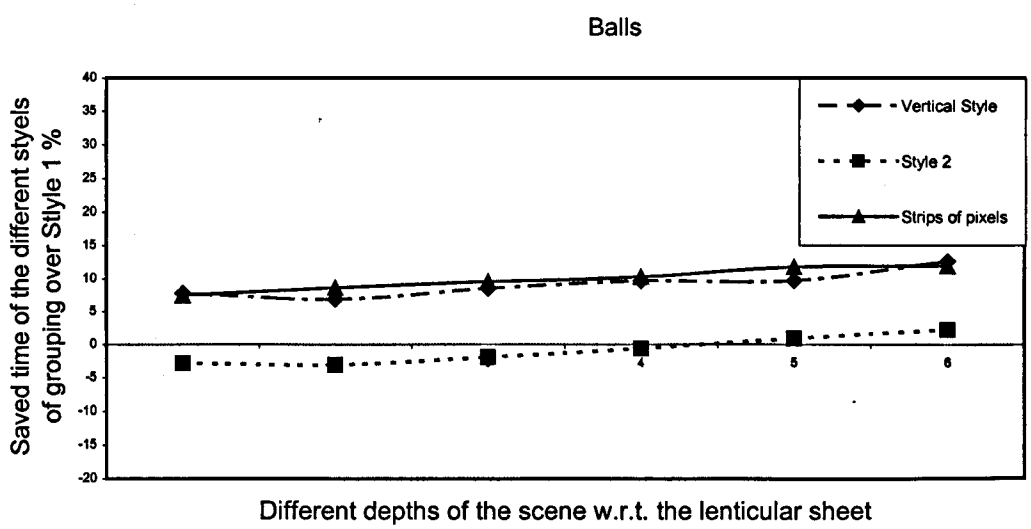


Figure 5.20(a): The improvement of time over style 1 for other styles of tracing in percentage for balls scene.

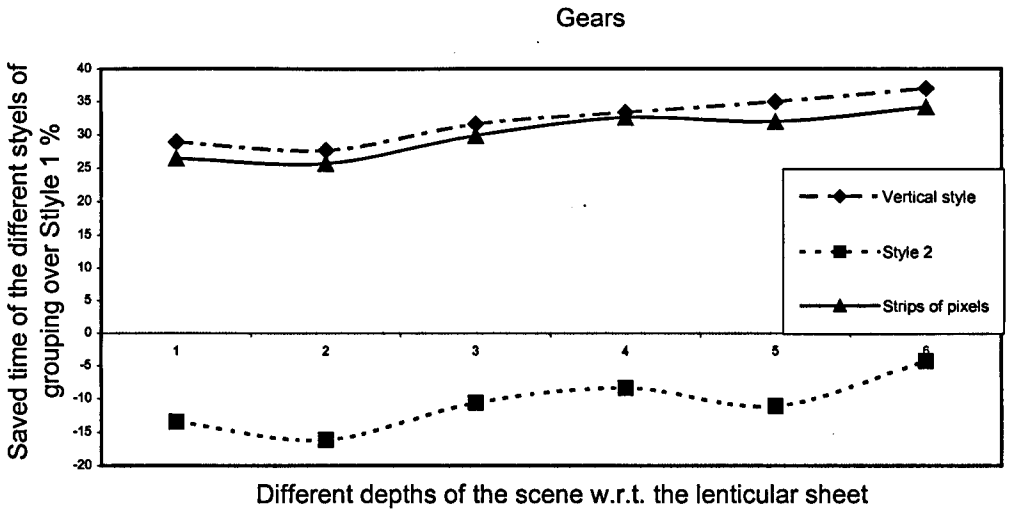


Figure 5.20(b): The improvement of time over style 1 for other styles of tracing in percentage for gears scene.

5.7.2 Micro-lens array

Code profiling is also used in the case of micro-lens array. Table 5.5 shows the number of times *shadow_intersect_objects* is executed, the percentage of time that is used by it, and the total time in seconds, which is spent in order to render the scene.

Depth	*Number of executions of shadow_intersect_objects*	*Time of the executions%*	*Total time for rendering*
1	14102496	57.9	30.1
2	14186037	58.3	31.2
3	14505484	59.6	33.3
4	14824933	60.9	35.3
5	15164184	62.3	37.9
6	15511446	63.7	41.1

Table 5.5(a): Shadow calculations for balls scene.

Depth	*Number of executions of shadow_intersect_objects*	*Time of the executions%*	*Total time for rendering*
1	23215820	75.3	124.1
2	23401674	75.9	125.1
3	24030382	77.9	135.3
4	24659090	80	142.1
5	25621352	83.1	148.1
6	26743028	86.7	156.1

Table 5.5(b): Shadow calculations for gears scene.

For micro-lens array, shadow cache reduces the number of executions of *shadow_intersect_objects* function. Tables 5.6(a-c) and 5.4(d-f) show the result of applying shadow cache to the balls and the gears scenes respectively. For the balls scene, shadow cache saves up to *16.1%, 18.6%,* and *17.7%* of the executions of the function in styles 1, 2, and tracing squares of pixels. For gears scene, it saves up to *10.7%, 8%,* and *6.2%* of the executions of the function in styles 1, 2, and tracing squares of pixels respectively. The tables also show in percentage the saved time due to applying shadow cache and the total time of rendering in seconds for each of the different styles.

Depth	*Number of executions of shadow_intersect_objects*	*Saved executions%*	*Saved time%*	*Total time for rendering*
1	11859802	15.9	4.3	28.8
2	11936771	15.9	4.8	29.7
3	12191145	16	6.3	31.2
4	12460536	15.9	7.1	32.8
5	12735676	16	9	34.5
6	13012729	16.1	11.5	36.4

Table 5.6(a): Style 1: saved shadow calculations for balls scene.

Depth	*Number of executions of shadow_intersect_objects*	*Saved executions%*	*Saved time%*	*Total time for rendering*
1	11504033	18.4	2.3	29.4
2	11543163	18.6	5.5	29.5
3	12208362	15.8	6.3	31.2
4	12795302	13.7	7.4	32.7
5	13460502	11.2	9.3	34.4
6	14164830	8.7	12	36.2

Table 5.6(b): Style 2: saved shadow calculations for balls scene.

Depth	*Number of executions of shadow_intersect_objects*	*Saved executions%*	*Saved time%*	*Total time for rendering*
1	11959179	15.2	2.3	29.4
2	12050835	15.1	4.8	29.7
3	12169233	16.1	6.6	31.1
4	12306143	17	8.5	32.3
5	12473491	17.7	10.1	34.1
6	12660814	17.6	13.7	35.5

Table 5.6(c): Squares of pixels: saved shadow calculations for balls scene.

Depth	*Number of executions of shadow_intersect_objects*	*Saved executions%*	*Saved time%*	*Total time for rendering*
1	20756498	10.6	2.5	121
2	20925194	10.6	1.9	122.7
3	21469476	10.7	6.4	126.7
4	22335551	9.4	3.4	137.3
5	23277847	9.1	5.1	140.5
6	24275732	9.2	4.8	148.7

Table 5.6(d): Style 1: saved shadow calculations for gears scene.

Depth	*Number of executions of shadow_intersect_objects*	*Saved executions%*	*Saved time%*	*Total time for rendering*
1	23280530	-0.3	-4.2	129.3
2	23163256	1	-6	132.6
3	24043096	-0.1	-1.5	137.3
4	24038577	2.5	1.3	140.3
5	24037024	6.2	1.4	146
6	24614099	8	0.7	155.1

Table 5.6(e): Style 2: saved shadow calculations for gears scene.

Depth	*Number of executions of shadow_intersect_objects*	*Saved executions%*	*Saved time%*	*Total time for rendering*
1	22038182	5.1	1.4	122.4
2	22516921	3.8	-3	128.9
3	23044949	4.1	2.7	131.6
4	23610235	4.3	3	137.8
5	24185198	5.6	0.8	146.9
6	24693449	7.7	0.3	155.6

Table 5.4(f): Squares of pixels: saved shadow calculations for gears scene.

Comparing tracing of similar pixels as well as tracing squares of pixels with style 1 shows that they do not achieve any improvement over style 1. See figure 5.21. Tracing of squares of pixels does not achieve improvement because of the randomness of the projected shadows in the tested scenes.

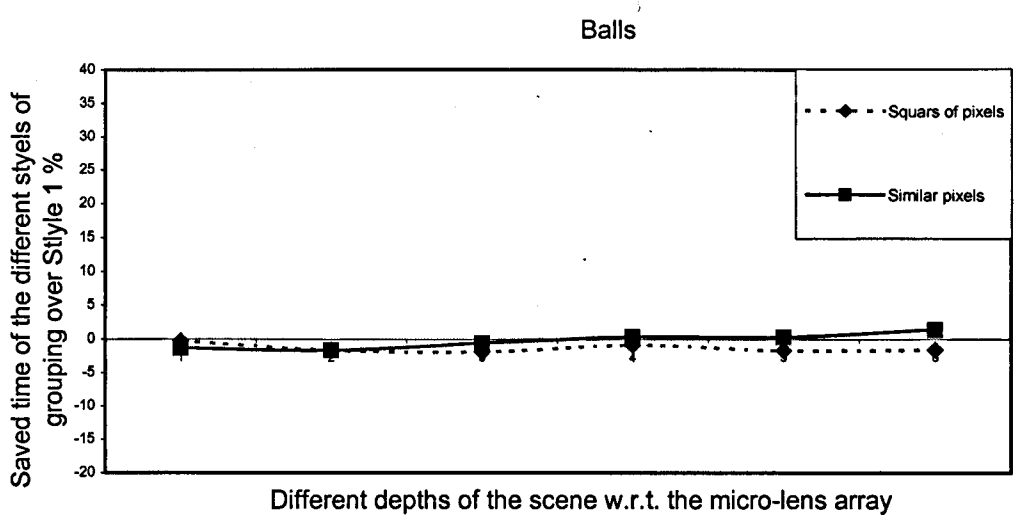


Figure 5.21(a): The improvement of time over style 1 for other styles of tracing in percentage for balls scene

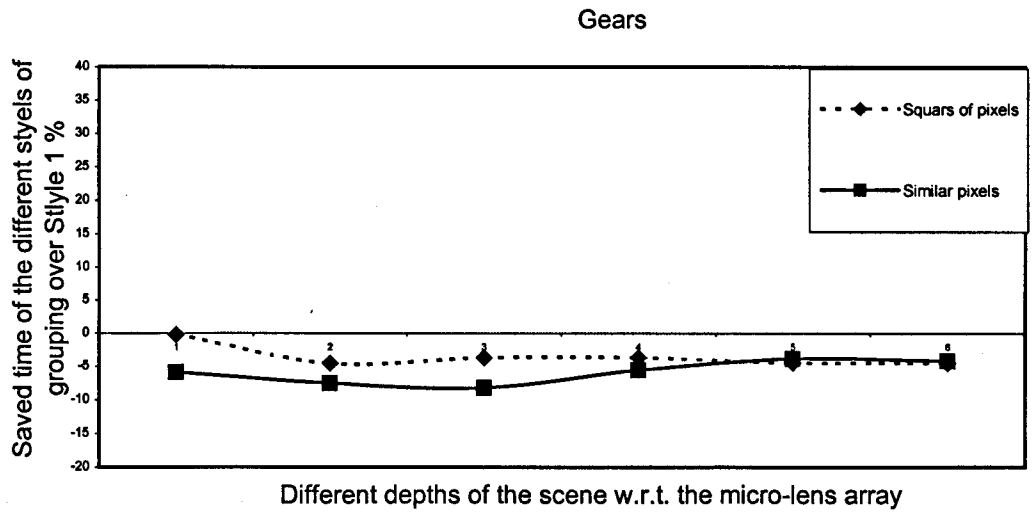


Figure 5.21(b): The improvement of time over style 1 for other styles of tracing in percentage for gears scene

5.8 Shadow cache and parallel integral ray tracing

5.8.1 Parallel ray tracing

Ray tracing is especially well suited for parallelism. Each pixel in the ray-traced image can be calculated independently of the rest of the pixels in the image. This property is known as data parallelism. Ray tracing is often placed in the category of the algorithms that are referred to as *embarrassingly parallel.* Although naïve ray tracing systems are trivial to implement in parallel. There are significant challenges involved in implementing an efficient photo-realistic parallel ray tracing system. Many of the algorithmic efficiency schemes developed to increase ray tracing time-performance work against the scalability of a parallel ray tracing system. Load balance, and constraints placed on the accessibility of data in shared memory present problems that must be overcome in a high time-performance ray tracing system. Many procedural textures can be designed and implemented with little difficulty in a concurrent execution environment. Problematic cases exist when texturing algorithms reference extensive modifiable data structures. In a shared memory environment such algorithms must use mutual exclusion locks to avoid memory corruption. Excessive use of mutual exclusion locks can severely degrade performance in situations where many processors are competing for the use of a shared resource [Ston98].

5.8.1 Shadow cache and parallelism

Since the ray tracing algorithm processes pixels independently of each other, a wide variety of problem decompositions are available for scheduling work on processors in a parallel computer. The choice of decomposition, as different styles of tracing, affects various aspects of ray tracing performance. Decompositions that retain some degree of ray coherence help shadow cache that uses caching of previous results to improve ray tracing performance. Decompositions that work in larger blocks help improve I/O and communication performance. Scanline cyclic decompositions like styles 1 and 3, improve communications. Block cyclic decompositions like tracing strips of pixels and tracing squares of pixels, have the best ray coherency, but tend to leave irregular sized areas at image borders, which require special handling. Block cyclic decompositions

also cause I/O routines to perform multiple seek operations when outputting completed pixel blocks. Pixel-cyclic decompositions such as style 2, provide the maximum degree of concurrency and scalability, but severely degrade ray coherency and add significantly to I/O and message passing overhead. All three of these problem decompositions have their own advantages and drawbacks, depending on the factors under consideration.

5.8.1.1 Multithreaded ray tracing

Multiprocessor computers utilising shared memory have become increasingly popular in recent years. As shared memory multiprocessors have developed, so have the methods for programming them. One of programming paradigms, which has come into widespread use, is known as multithreaded programming. A thread is a context of execution within a process. A traditional process only has a single thread of execution. Many modern operating systems have the capability of supporting multiple threads of execution within a process. The threads within a process can share resources owned by that process such as memory and file descriptors. A uniprocessor computer executes threads within a process using a time-sharing scheduling system similar to the ones used for scheduling execution of entire processes. A multiprocessor can execute several threads concurrently. Multithreaded programming can improve execution speed of some applications even when running on a uniprocessor system by performing useful work while other threads are blocked in operating system calls doing I/O or similar tasks. The challenge offered by multithreaded programming is in controlling the use of resources, which are shared by multiple concurrently, executing threads. Sharing of read-only data is accomplished trivially, since no modifications are being made to the data, it is safe for all threads to access data structures concurrently. Sharing of writable data must be controlled through the use of mutual exclusion locks and condition variables. In order to write multithreaded programs, a target system must provide operating system services and libraries for the creation and management of threads and their resources. Thread interfaces in widespread use today include POSIX threads, and Unix International threads. The most basic services required for implementation of multi-threaded programs are thread creation, thread cancellation, mutual exclusion locks, and condition variables [Ston98].

Many of the standard ray tracing algorithms may be successfully implemented in a multithreaded environment if care is given to their design. The design decisions involved in adapting standard ray tracing algorithms for multithreading centre on management of resources which are shared by multiple threads. In ray tracing, shared resources typically include the object database, texture structures, images used by texturing procedures, volumetric data, run-time parameters and settings, and output image storage. Depending on how these shared resources are used during rendering, they may present problems in a multithreaded environment. Careful design decisions may avoid many of these potential problems. Data structures, which may present problems in a multithreaded ray tracing, are those, which are modified during rendering. Structures, which are fully constructed prior to creation of child threads, are ideal. Once multithreading begins it is advantageous for structures to be treated as read-only data. Structures, which are considered read-only do not need to be protected with mutual exclusion locks or other access synchronization primitives. Modifiable structures need delicate handling, requiring access synchronization in order to prevent data corruption. Every time a modifiable structure is accessed, a mutual exclusion lock must be granted before a thread may read or modify the structure. Without access synchronisation, one thread may modify data while another thread is reading the same data. Modern shared memory multiprocessors do not guarantee that memory accesses are atomic. An example of this would be two threads accessing a double precision floating point variable concurrently. If one thread is reading and the other thread is writing, it is possible for the reader thread to read some bytes from the original value in the variable and some bytes from the value that is being written concurrently. The resulting value read is then corrupted. In multithreaded ray tracing shared structures should be treated as read-only objects at rendering time, allowing all threads to access objects concurrently without the need for mutual exclusion locks or other access synchronization controls. Dynamic data structures, which must be modified during rendering, may be handled in one of two ways. Structures of large size may be shared through the use of mutual exclusion locks or similar access synchronization techniques [Stone98]. Objects, which are small or are only meaningful in the context of one thread of execution, may be implemented in a thread specific memory area, which is the

category shadow cache falls in where each thread is responsible for a group of pixels. In style 1 each thread is responsible for a number of rows while in style 3 each thread is responsible for a number of columns. Style 2 yields that each thread is assigned to a group of similar pixels, for example the third pixel in each lens in every row. In block decompositions cases such as strips of pixels or squares of pixels, block of pixels are distributed on the working threads in an alternating manner. In this sense each thread has its own shadow cache and does not need to access or modify any shared data. This keeps the execution of each thread free of any interruption and supports the scalability over any number of processors.

5.9 Summary

This chapter explains an analysis of the relationship between rays and shadows, which is established in order to find out the optimal pixel-tracing style for integral ray tracing that leads to the fastest shadow intersection tests using shadow cache algorithm, and the novel styles developed.

Shadow cache works by keeping a cache of the most recent shadow-casting object for each light source. Ray-Object coherence is exploited to decrease the number of shadow intersection tests. If an object prevents a light source from contribution to a pixel's intensity, then the same object is likely to block the same light for an adjacent pixel.

In order to apply shadow cache to integral ray tracing, it requires coherent grouping of pixels that ensures successively tracing of pixels whose primary rays share spatial coherence.

Cache Calculations are illustrated and showed that tracing pixels in a non-coherent way is very computationally expensive and could duplicate the computation cost of shadow rays in shadow cache algorithm. The structure of the lenses and the camera model in the integral ray tracing affects the way primary rays are spawned as well as the spatial coherence among them. In order to find out the optimal pixel-tracing style that attains

the maximum number of continuous cache hits, the relationship between pixels and their primary rays and shadows is analysed.

The analysis consists of the way primary rays relate to the shadow size and location and its affect on the number of continuous cache hits. Three main styles of tracing pixels for lenticular sheets are developed; Style 1: Tracing of consecutive pixels in the horizontal direction, which is the normal style and two novel styles style 2: Tracing similar pixels in adjacent lenses, and style 3:Tracing pixels in the vertical direction. For micro-lens array there is two main styles; Tracing of consecutive pixels which is the normal style, and the novel pixel-tracing of similar pixels in adjacent lenses.

Studying the different styles of tracing in order to find the optimum style for maximum number of continuous cache hits yields that missing a cache hit even once during tracing pixels can be very expensive. If there are a number of pixels whose rays intersect points in shadow, the optimum style of pixel tracing is the one, which traces all those pixels successively before switching to other pixels. This is achieved by merging different styles into new ones that work on pixels in two dimensions.

Two-dimensional style for lenticular sheet traces strips of pixels that cover the maximum area of the projected shadow. For micro-lens array, square areas of pixels are traced in order to cover the maximum area of the projected shadow and accommodate the ray-object coherence.

The results show that the coherent pixel-tracing styles developed with respect to the modelled integral camera resulted in time-improving integral ray tracing by up to 41% for lenticular sheets and 18.6% for micro-lens arrays.

Running integral ray tracing on parallel computers needs some careful manipulation of the cached object in order to avoid using mutual exclusion locks that severely degrade the time-performance. In this sense each thread has its own shadow cache and does not

need to access or modify any shared data. This keeps the execution of each thread free of any interruption and supports the scalability over any number of processors.

Next chapter explain a novel algorithm for accelerating integral ray tracing using reprojection method in order to generate full integral frames of the rendered scene and accelerate integral ray tracing by four times more than its normal execution speed.

Chapter 6

Rapid Integral Ray Tracing Using Reprojection

The idea of reprojection is to avoid tracing pixels and reuse results from the previous frame for generating the new frame in order to reduce the large number of intersection tests associated with tracing pixels in each frame. The novel integral reprojection algorithm proposed accelerates integral ray tracing by four times more than its normal execution speed. This improves in terms of time the generation of sequence of integral images. The algorithm developed treats the lenticular sub-images of a frame separately and uses reprojection to generate new sub-images with information leftover from the correspondent sub-images in the frame immediately before it. Visual artefacts caused by reprojection are treated using z-buffered projection and interpolation while only missed pixels are ray-traced again. The integral reprojection process is illustrated in figure 6.1.

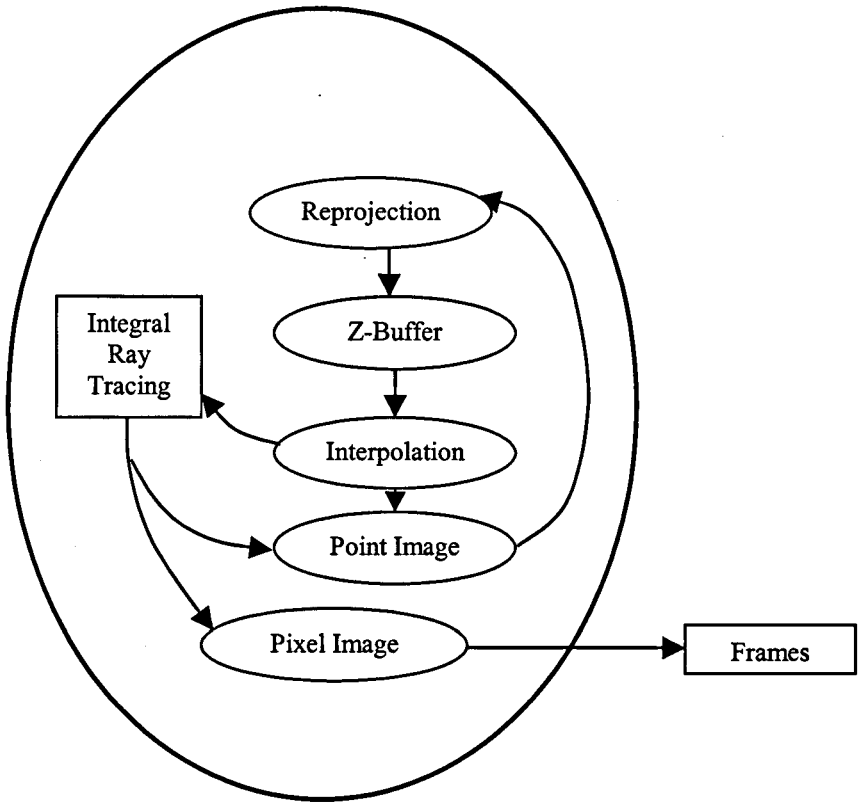


Figure 6.1: Rapid integral ray tracing using reprojection.

6.1 Projection in integral ray tracing

In the recording stage of ray tracing algorithm, rays that sample the scene are used to find intersection points with objects in the scene. The intersection points are projected onto the plane, which contains the image. This plane is termed as the projection plane. The output projections of this process define the pixels of the image. The projection process can be represented by matrix multiplication of camera matrix C and the 3D world coordinates $(X, Y, Z, 1)$ of the intersection point, which being projected onto the projection plane:

$$\begin{bmatrix} wx \\ wy \\ wz \\ w \end{bmatrix} = C \begin{bmatrix} X \\ Y \\ Z \\ 1 \end{bmatrix} \tag{6.1}$$

where (x, y, z) is the image coordinates of the projection and w is a scaling factor. (x, y) is the pixel position in the image. The projection plane is z distance away from the centre of projection and its constant for all the pixels. Camera matrix C will be explained in the next section.

6.1.1 Camera matrix

Graphics renderers that use pinhole camera use perspective projection in their cameras. Perspective projection makes projections of far objects to be smaller than those of closer objects. Camera matrix consists of projection matrix and transformation matrices that transform the scene from the world coordinate to coordinate of the viewing camera. Transformation matrices will be described later in this chapter.

If there is a projection plane at distance d from the centre of projection of the camera which represents the origin of the camera coordinates, and normal to the z axis. P is a point to be projected onto the projection plane. Then in order to calculate $P_p = (x_p, y_p, z_p)$ the perspective projection of P, similar triangles are used as shown in figure 6.3 to establish the ratios:

$$x_p/d = x/z; \qquad y_p/d = y/z \tag{6.2}$$

Multiplying each side by d yields:

$$x_p = x/(z/d), \qquad y_p = y/(z/d) \tag{6.3}$$

The distance d is simply a scale factor applied to x_p and y_p. This mathematical operations can be represented by the projection matrix, which is a 4 × 4 matrix expressed as follows:

$$M = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \tag{6.4}$$

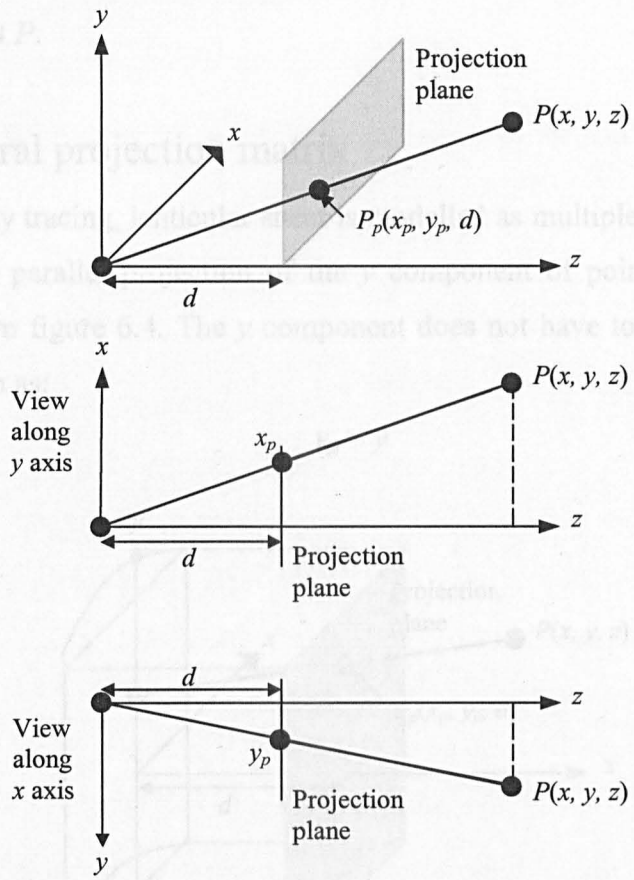


Figure 6.3: Perspective projection.

Multiplying the point $P = \begin{bmatrix} x & y & z & 1 \end{bmatrix}^T$ by the matrix M yields the general homogeneous point $\begin{bmatrix} X & Y & Z & W \end{bmatrix}^T$, where:

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = M \cdot P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{6.5}$$

In equation 5.5, z/d has defined as W. Dividing $\begin{bmatrix} X & Y & Z & W \end{bmatrix}^T$ by W and dropping the fourth coordinate yields:

$$(X/W,\ Y/W,\ Z/W) = (x_p,\ y_p,\ z_p) = (x/(z/d),\ y/(z/d),\ d) \tag{6.6}$$

These equations are the results of equation 6.2 and define the position of the pixel of the projected point P.

6.1.2 Integral projection matrix

For integral ray tracing, lenticular sheet is modelled as multiple axial camera and yields an orthogonal parallel projection of the y component of point P onto the projection plane as shown figure 6.4. The y component does not have to be scaled by d/z and y_p may be written as:

$$y_p = y \tag{6.7}$$

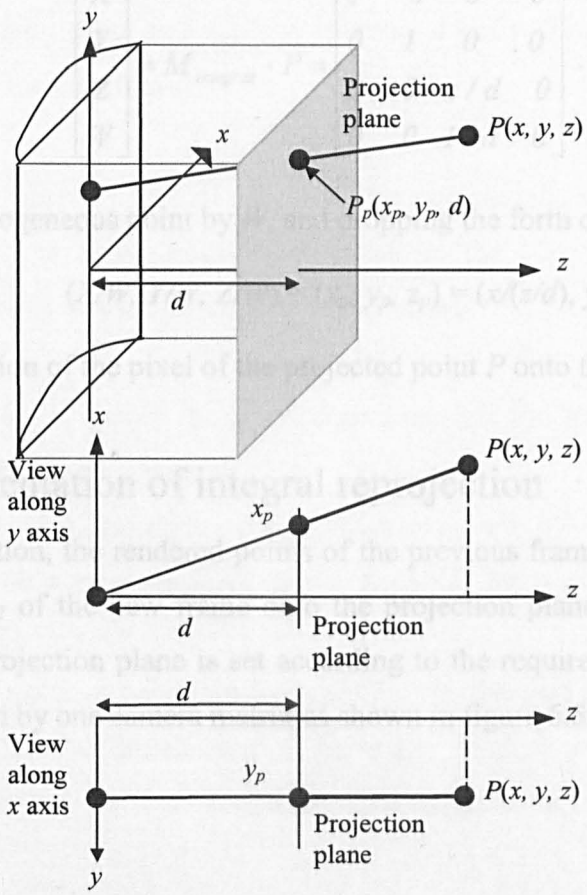


Figure 6.4: Projection for modelled integral cylindrical lens.

This yields to a new projection matrix termed as *integral projection matrix* $M_{integral}$, which can be expressed as:

$$M_{integral} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & z/d & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \tag{6.8}$$

Integral projection matrix is one of two components of the camera matrix in integral ray tracing.

Multiplying the point $P = [x \quad y \quad z \quad 1]^T$ by the matrix $M_{integral}$ yields the general homogeneous point $[X \quad Y \quad Z \quad W]^T$, where:

$$\begin{bmatrix} X \\ Y \\ Z \\ W \end{bmatrix} = M_{integral} \cdot P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & z/d & 0 \\ 0 & 0 & 1/d & 0 \end{bmatrix} \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{6.9}$$

Dividing the homogeneous point by W, and dropping the forth coordinate yields:

$$(X/W, Y/W, Z/W) = (x_p, y_p, z_p) = (x/(z/d), y, d) \tag{6.10}$$

which is the position of the pixel of the projected point P onto the projection plane.

6.1.3 Implementation of integral reprojection

In integral projection, the rendered points of the previous frame is to be reprojected via camera matrix C_2 of the new frame onto the projection plane of the new frame. The position of the projection plane is set according to the required view. Each cylindrical lens is represented by one camera matrix as shown in figure 6.5.

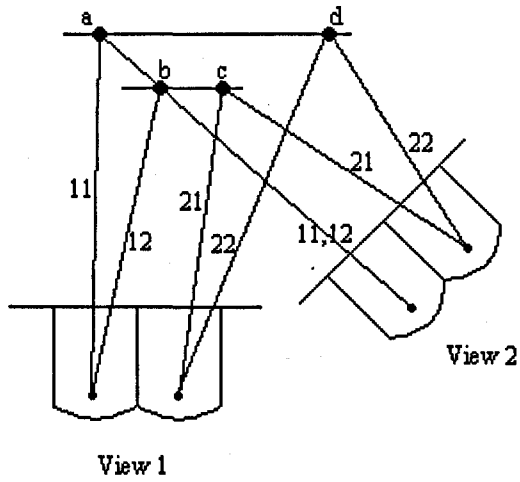
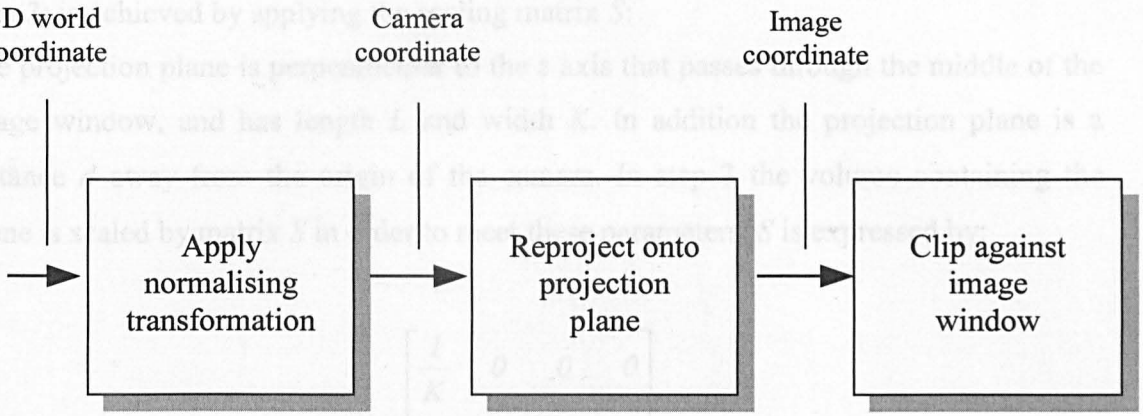


Figure 6.5: Integral reprojection

This simple process saves enormous amount of ray tracing computations with their associated intersection tests and shading operations and radically reduces the rendering execution time of the new frame.

The process by which the pixels of the new frame are calculated is summarized in figure 6.6. The first step is to normalise and transform the scene to the viewing camera coordinate. The series of transformations is as follows:

1. Translate the origin of the scene coordinate via the translation matrix T to the origin of the viewing camera.
2. Rotate the scene coordinate via the rotation matrix R such that the normal vector becomes the z axis, the right vector becomes the x axis, and the upper vector becomes the y axis.
3. Scale the volume containing the scene via the scaling matrix S in order to accommodate the dimensions of the image window and the distance by which the projection plane is separated from the origin of the camera.



6.6: Implementation of 3D viewing

Step 1: is achieved by applying the translation matrix T:

$$T = \begin{bmatrix} 1 & 0 & 0 & -c_x \\ 0 & 1 & 0 & -c_y \\ 0 & 0 & 1 & -c_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.11}$$

where (c_x, c_y, c_z) is the origin of the viewing camera.

Step 2: involves applying the rotation matrix R:

The row vectors to perform step 2 are the unit vectors along the x, y, and z axes [Fole90]. The scene is rotated onto the x, y, z axes of the camera, consequently

$$R = \begin{bmatrix} U_x & U_y & U_z & 0 \\ V_x & V_y & V_z & 0 \\ N_x & N_y & N_z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.12}$$

where U, V, and N are the unit vectors along the x, y, and z axes respectively.

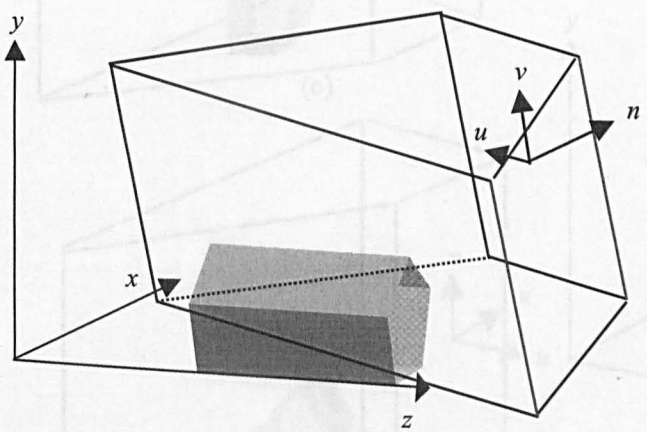
Step 3: is achieved by applying the scaling matrix S:

The projection plane is perpendicular to the z axis that passes through the middle of the image window, and has length L and width K. In addition the projection plane is a distance d away from the origin of the camera. In step 3 the volume containing the scene is scaled by matrix S in order to meet these parameters. S is expressed by:

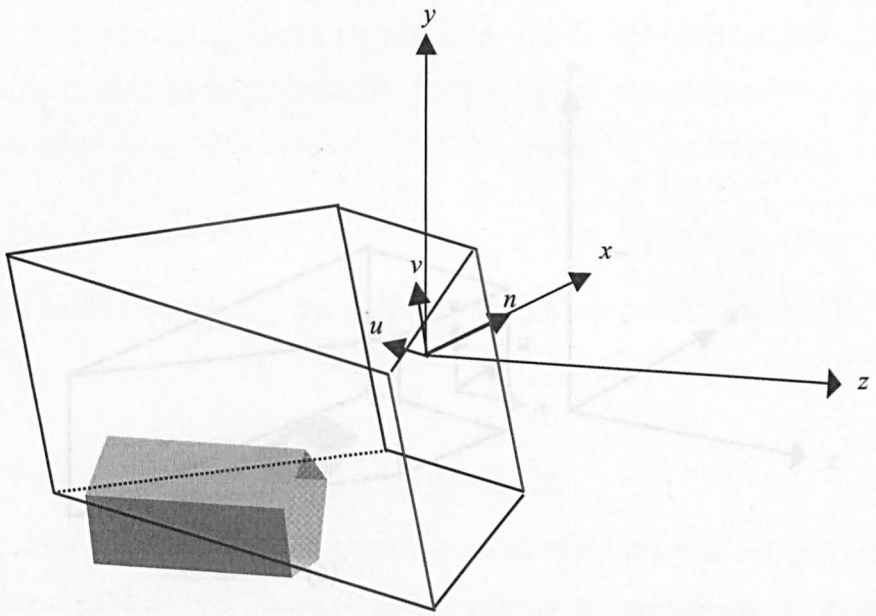
$$S = \begin{bmatrix} \frac{1}{K} & 0 & 0 & 0 \\ 0 & \frac{1}{L} & 0 & 0 \\ 0 & 0 & \frac{1}{d} & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{6.13}$$

Figure 6.7 shows the series of transformations as applied to a scene prior to integral projection. Multiplying the three matrices produces the *normalisation transformation matrix N*:

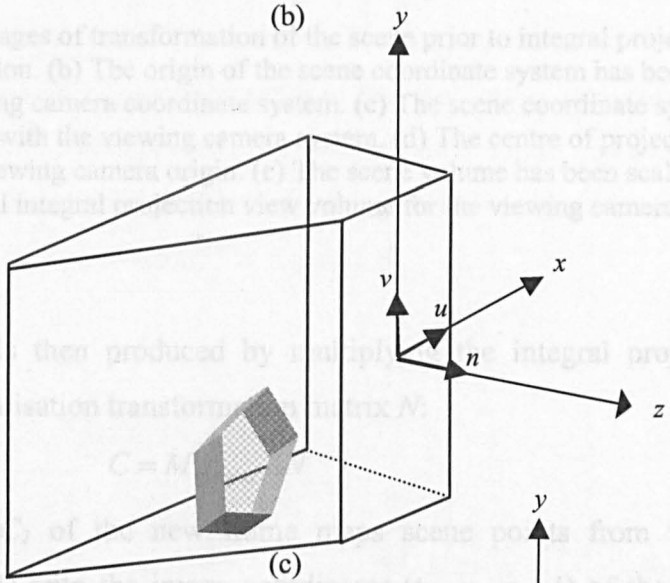
$$N = S \cdot R \cdot T \tag{6.14}$$



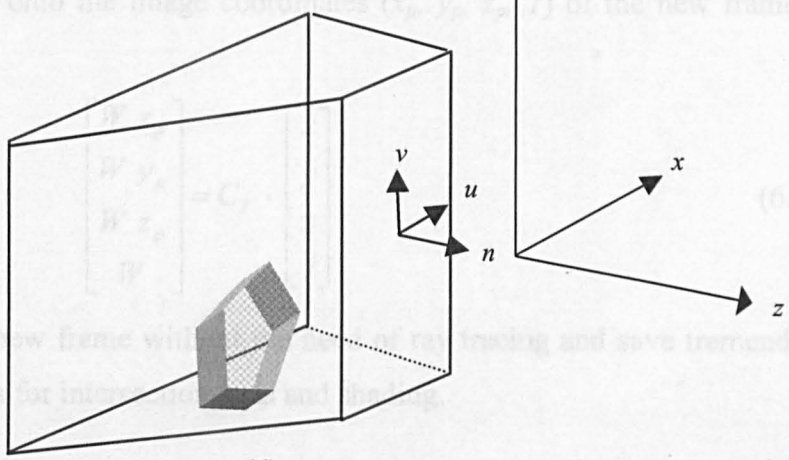
(a)



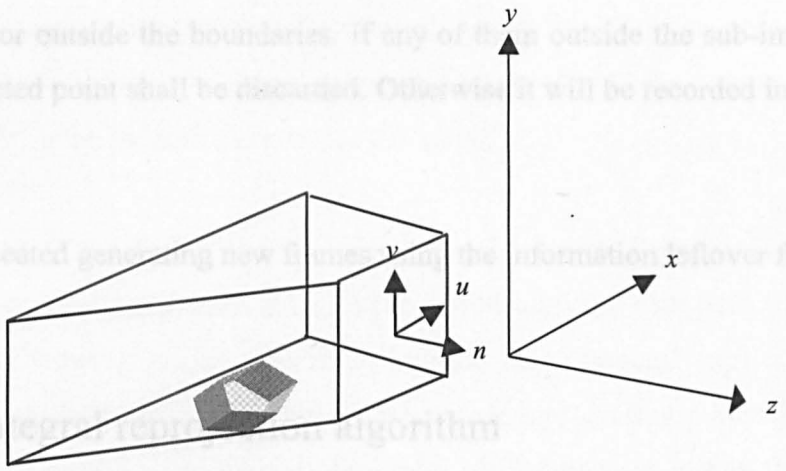
(b)



(c)



(d)



(e)

Figure 6.7: Various stages of transformation of the scene prior to integral projection. (a) The original viewing situation. (b) The origin of the scene coordinate system has been translated to the origin of the viewing camera coordinate system. (c) The scene coordinate system has been rotated to be aligned with the viewing camera system. (d) The centre of projection has been translated to the viewing camera origin. (e) The scene volume has been scaled onto the canonical integral projection view volume for the viewing camera.

The camera matrix is then produced by multiplying the integral projection matrix $M_{integral}$ and the normalisation transformation matrix N:

$$C = M_{integral} \cdot N \tag{6.15}$$

The camera matrix C_2 of the new frame maps scene points from the 3D world coordinates (x, y, z, 1) onto the image coordinates (x_p, y_p, z_p, 1) of the new frame as follows:

$$\begin{bmatrix} W\,x_p \\ W\,y_p \\ W\,z_p \\ W \end{bmatrix} = C_2 \cdot \begin{bmatrix} x \\ y \\ z \\ 1 \end{bmatrix} \tag{6.16}$$

This will generate the new frame without the need of ray tracing and save tremendous amount of computations for intersection tests and shading.

The final step is to check whether x_p and y_p are within the sub-image boundaries for the corresponding camera, or outside the boundaries. If any of them outside the sub-image boundaries, the reprojected point shall be discarded. Otherwise it will be recorded in the new frame.

The same process is repeated generating new frames using the information leftover from previous frames.

6.1.4 The novel integral reprojection algorithm

The reprojection process requires certain information about every pixel in the previous frame in order to generate the current frame. Such information as the location of a point, object ID, and colour are required for each intersection point in order to accomplish the reprojection and produce the associated pixel in the new frame. Integral ray tracing has been modified in order to generate two two-dimensional arrays representing the *pixel image* and the *point image* respectively. Elements of both arrays are simply linked by their position in the arrays as illustrated in figure 6.8.

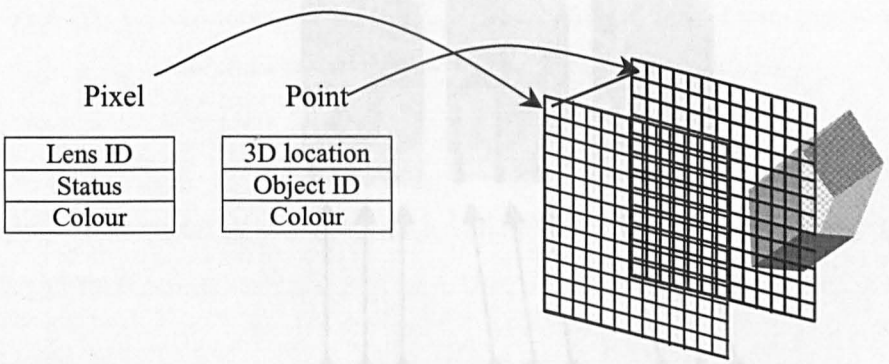


Figure 6.8: Pixel image and Point Image.

Each element of the point image consists of: the location of an intersection point, the ID of the intersected object, and the associated colour information. Each element of the

pixel image consists of: the ID of the associated lens, the colour of the pixel, and status. The status of a pixel represents the status of the colour and it can be represented by three conditions: (1) '0' the colour of a freshly generated point, (2) '1' the colour of a reprojected point, (3) '-1' no point is projected onto this pixel.

Obviously the first frame of the sequence is fully ray-traced prior to the points being reprojected using the previously explained integral projection onto the new next frame with respect to the new viewing position. Each cylindrical lens (camera) applies the reprojection algorithm separately from one another. Therefore reprojecting points from the previous frame via the same camera generates the new sub image as shown in figure 6.9.

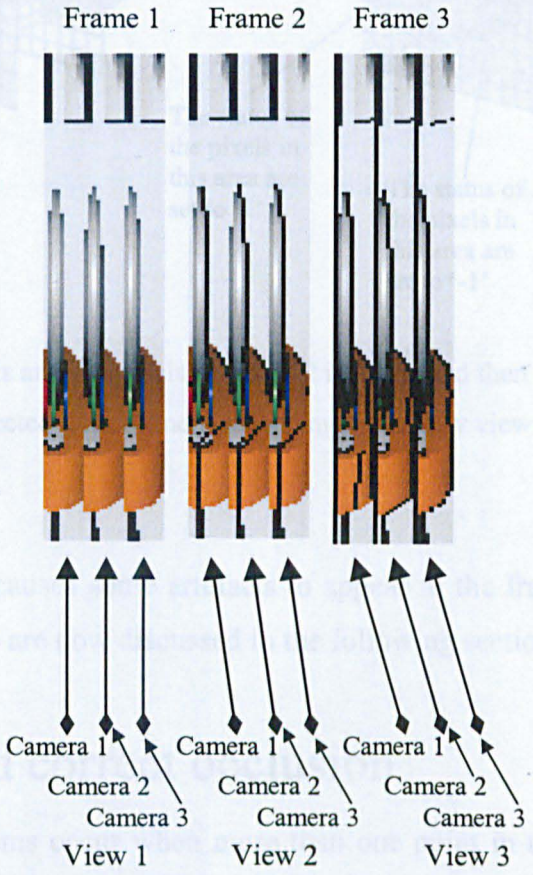


Figure 6.9: Each sub image of frame 2 is generated by reprojecting points of the corresponding sub image in frame 1 through its associated camera. The same for frame 3.

Multiplying each point P from the previous frame by the new camera matrix of the new frame yields the pixel location in the new frame's coordinate. Therefore information of the point P is stored in this location in the new point image of the current frame. The colour of the point is then stored in the corresponding pixel in the pixel image with the status set to '1'. The new point image is first initialised by setting the colour to black and by setting status of the pixels in the pixel image to '-1' as shown in figure 6.10.

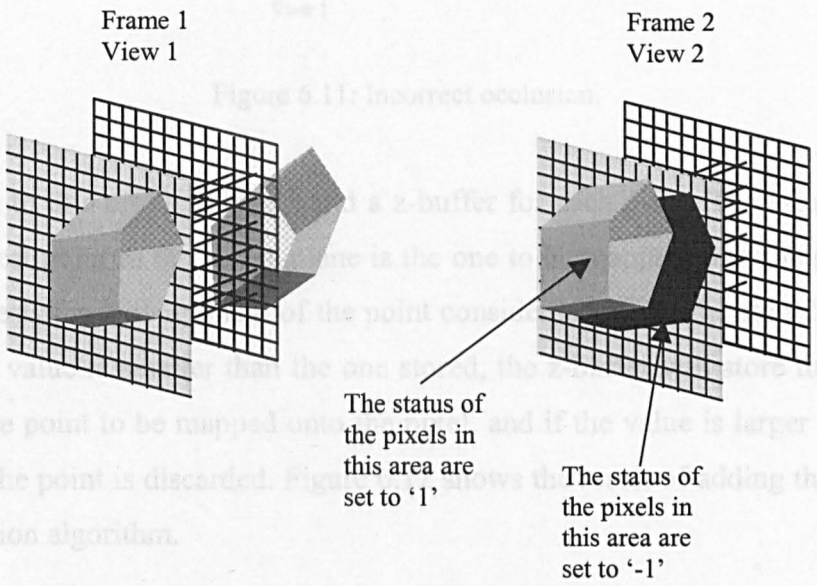


Figure 6.10: All the points and the pixels of frame 2 is initialised then points of frame1 is reprojected onto frame 2 according to the new view.

The reprojection process causes some artefacts to appear in the frames. These specific artefact and their solutions are now discussed in the following sections.

6.2 Z-buffer and correct occlusion

Incorrect occlusion problems occur when more than one point in the point image map onto the same pixel position in the new frame as shown in figure 6.11.

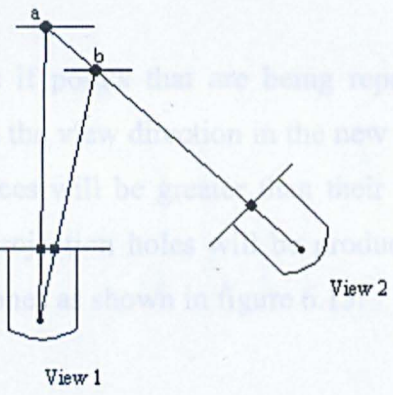


Figure 6.11: Incorrect occlusion.

The solution to this problem is to build a z-buffer for each pixel. The z-buffer ensures that the closest point to the image plane is the one to be mapped onto the pixel. The z-buffer first tests the z-component of the point considered against the stored value in the buffer if the value is smaller than the one stored, the z-buffer will store the new value and allow the point to be mapped onto the pixel, and if the value is larger than the one stored then the point is discarded. Figure 6.12 shows the result of adding the z-buffer to the reprojection algorithm.

(a) (b)

Figure 6.12: (a) Incorrect occlusion. (b) Correct occlusion due to the z-buffer.

6.3 Interpolation

The second problem appears if points that are being reprojected exist on a surface whose normal rotates towards the view direction in the new view. In the new frame, the projected areas of such surfaces will be greater than their projections in the previous frame and for one-to-one reprojection holes will be produced or background surfaces will appear through the front ones as shown in figure 6.13.

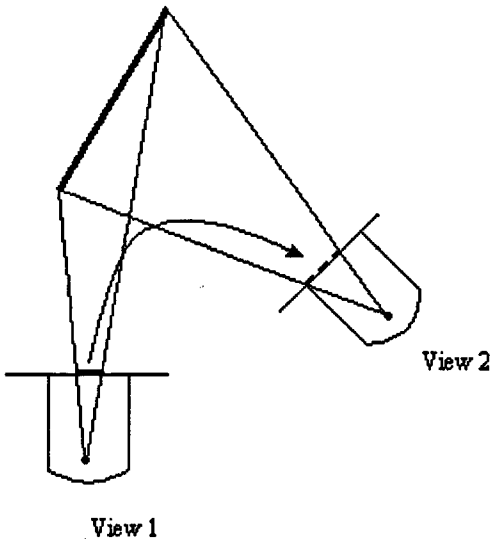


Figure 6.13: Holes in the reprojected area due to increasing its size.

The effect of this problem on an image is shown in figure 6.14. Note how holes caused by increasing the projected area of surfaces form coherent patterns. An interpolation algorithm has been developed to overcome this problem. The algorithm tests each pixel against neighbours and if an point is reprojected onto this pixel, and has a different object ID than the one of the neighbours therefore this pixel has a an incorrect point reprojected onto it. The solution is to set the colour of this pixel to the colour of the neighbour pixel. Figure 6.15 shows that coherent black patterns are removed after applying the interpolation algorithm.

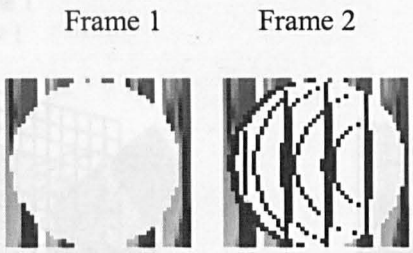


Figure 6.14: Holes caused by increasing the projected area of surfaces form coherent patterns.

Figure 6.15: Interpolation removes the holes caused by increasing the projected area.

6.4 Filling holes

Pixels with no points mapped onto them are called missed pixels and they appear as holes in the new frame. Missed pixels are ray-traced in order to generate their intersection points and produce their colour. Missed pixels are the result of: (1) Occluded areas in the previous frame that should appear in the new frame are missing. This is illustrated in figures 6.16 and 6.17. (2) The new frame shows an area that has not been covered by the previous frame. In example of this occurrence is shown in figure 6.18, which shows the upper left corner of figure 6.17.

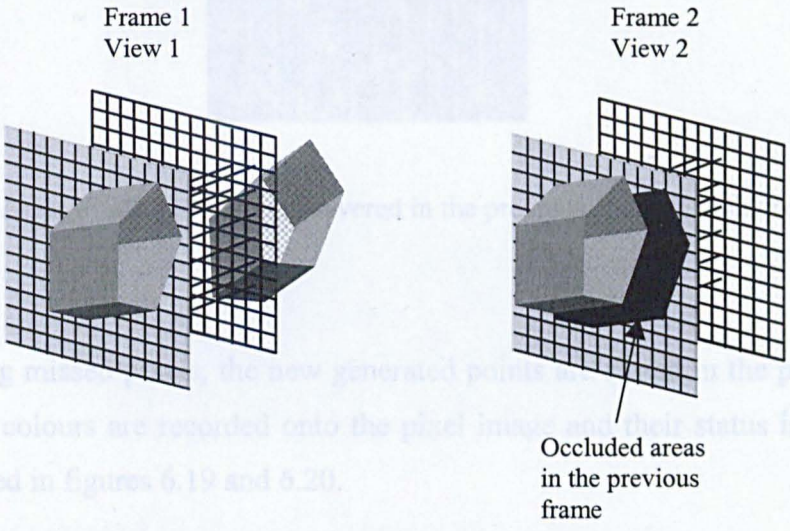


Figure 6.16: Occluded areas in the previous frame have no points mapped to them in the new frame.

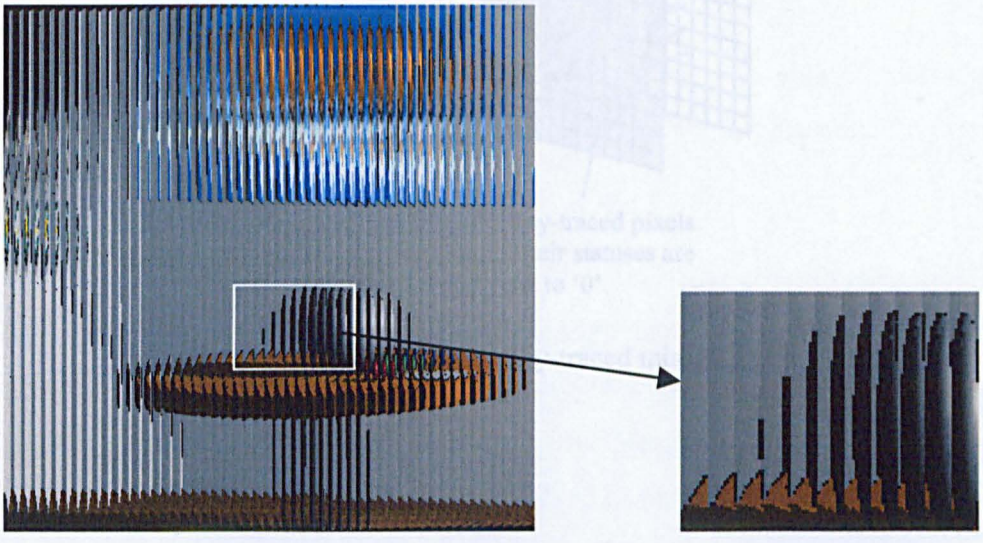


Figure 6.17: Occluded areas in the previous frame coloured black in the current frame.

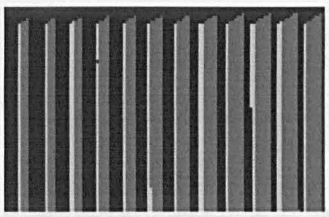


Figure 6.18: Area, which has not been covered in the previous frame, is coloured black.

After ray tracing missed pixels, the new generated points are stored in the point image. The associated colours are recorded onto the pixel image and their status is set to '0'. This is illustrated in figures 6.19 and 6.20.

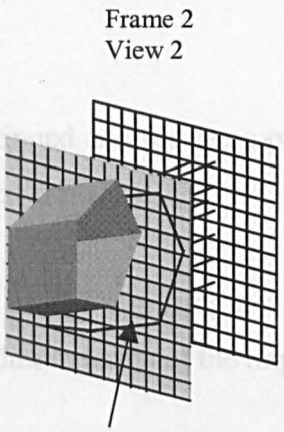


Figure 6.19: Ray-traced missed pixels.

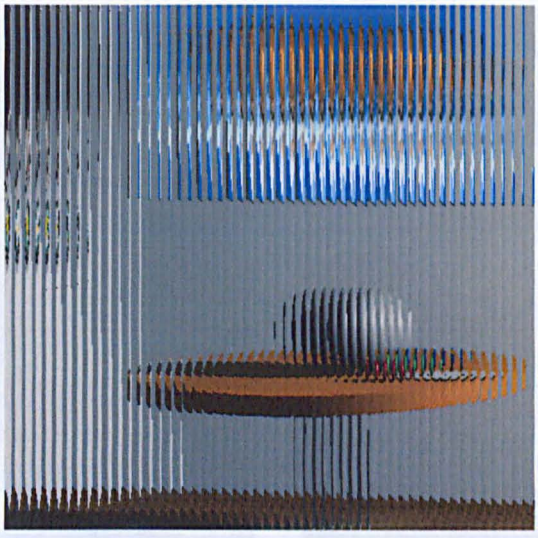


Figure 6.20: Figure 6.17 after ray trace missed pixels.

The recursive operation is then continued and the same process, using the points of the new frame is used to generate the next frame.

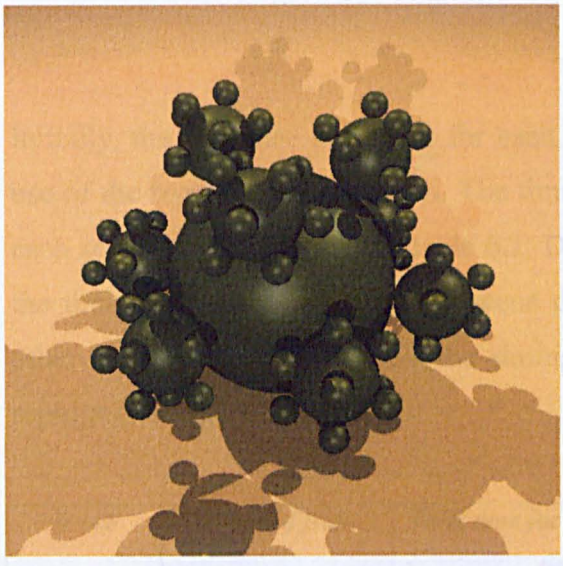
The next section describes the tests carried out using a number of scenes and discusses the results of the reprojection algorithm in terms of the improvement in processing time.

6.5 Tests and results

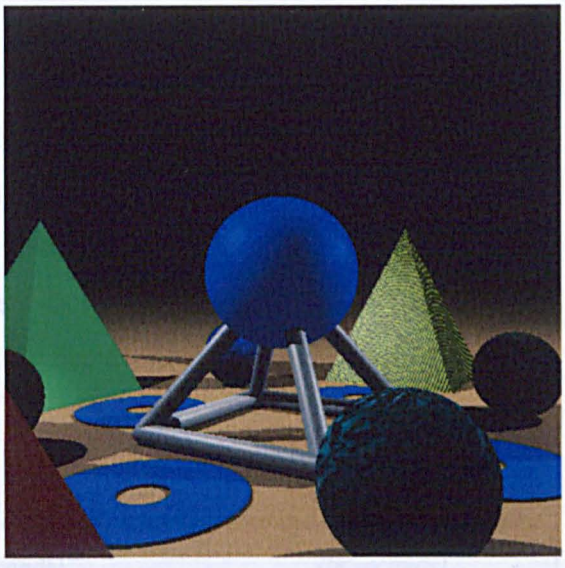
Tests were carried out using a walk-through camera on different scenes each of a different nature. The scenes used are shown in figure 6.21. The Experiments were conducted using a Compaq AlphaServer machine, which was described in chapter 5. The resolution of the frame was set to 512 × 512 pixels per frame.

(a) Teapot scene

(b) Room scene



(c) Small-balls scene



(d) Primitives scene

(e) Tree scene

Figure 6.20: Test scenes

Initially, the sequence of frames for each scene is fully integral ray-traced without the use of the reprojection algorithm. The time and the number of the generated frames for each scene are summarised in table 6.1. Due to the different complexity of each scene, the time needed for ray tracing a scene differs from the times needed for ray tracing other scenes. Table 6.2 shows the timings for the same frames generated using the reprojection algorithm.

Scene	*Number of frames*	*Total time for rendering the frames (in seconds)*	*Average time for each frame (in seconds)*
Teapot	655	2272	4.469
Room	680	1517	2.231
Small-balls	264	606	2.295
Primitives	1010	2950	2.921
Tree	264	24649	93.367

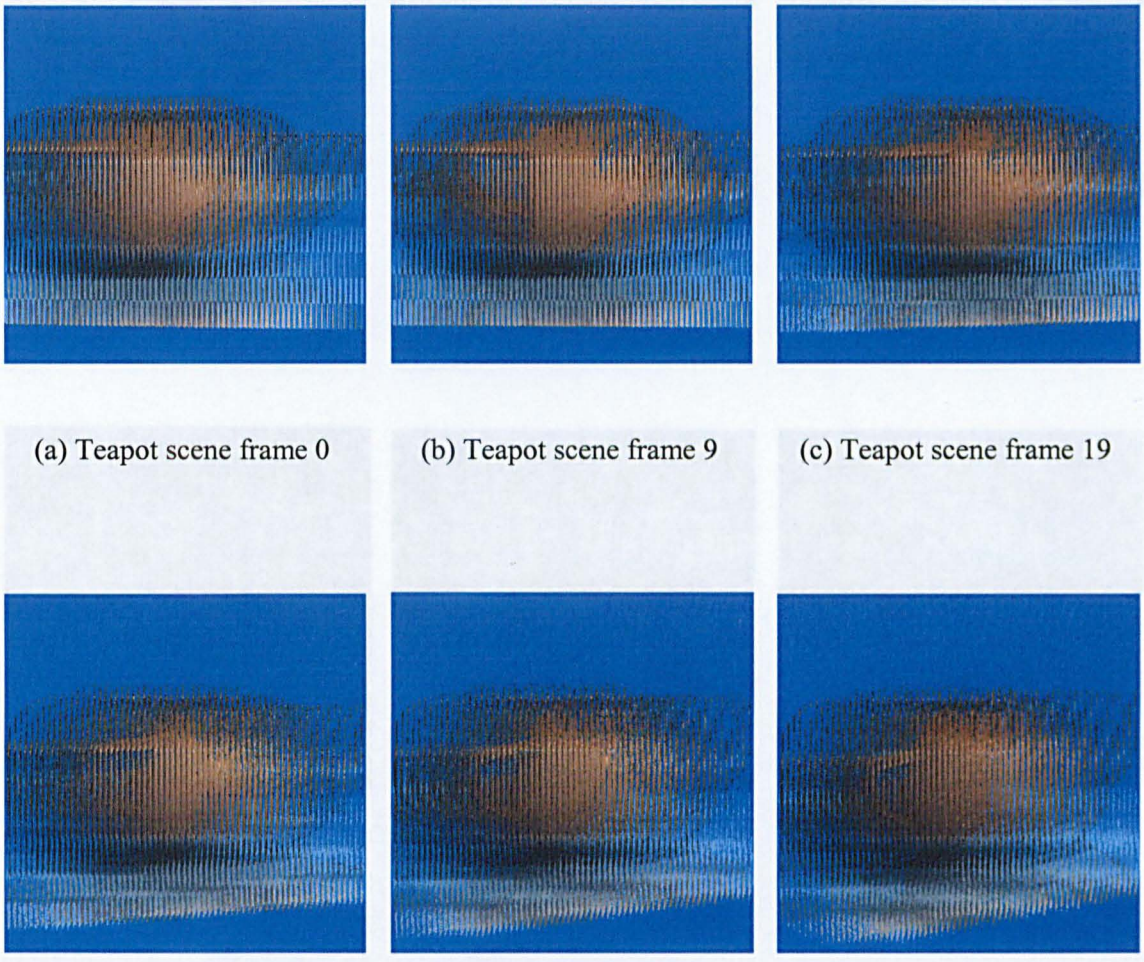
Table 6.1: Timings of fully integral ray-traced frames for test scenes.

Scene	*Total time for rendering the frames using reprojection (in seconds)*	*Time for reprojection (in seconds)*	*Time for ray tracing (in seconds)*	*Total saved time (in seconds)*
Teapot	1253	303	950	1019
Room	653	298	355	864
Small	292	120	172	314
Primitives	1290	584	706	1660
Tree	5993	120	5873	18656
Scene	*Average reprojection time per frame (in seconds)*	*Average number of saved rays per frame*	*Average time for rendering each frame using reprojection (in seconds)*	*Average saved time for ray tracing%*
Teapot	0.463	106756 (40.6%)	1.912	58.2%
Room	0.438	187734 (71.6%)	0.960	76.59%
Small-balls	0.455	192700 (73.5%)	1.106	71.6%
Primitives	0.578	176340 (67.3%)	1.277	76.1%
Tree	0.455	172899 (66%)	22.701	76.2%

Table 6.2: Timings for rendering the frames using reprojection.

The reader can see from the tables the large achievement in terms of time reduction that the new integral reprojection algorithm made. The unique reprojection algorithm saved about 75% of the ray tracing time for any kind of scene with any complexity and about 70% of the rays to be ray-traced in any frame. This made the generation of the frame sequence four times faster than its original execution speed before applying the integral reprojection algorithm. Integral reprojection consumes about half a second per frame independent of the scene nature and the time for ray tracing the scene. The approach adopted saved up to five hours and 11 minutes of computation using the AlphaServer machine for the Tree scene. Figures 6.21 to 6.25 show different frames of the test scenes

rendered using integral ray tracing with integral reprojection. Refer to the enclosed CD for videos generated from the frame sequences for each scene.

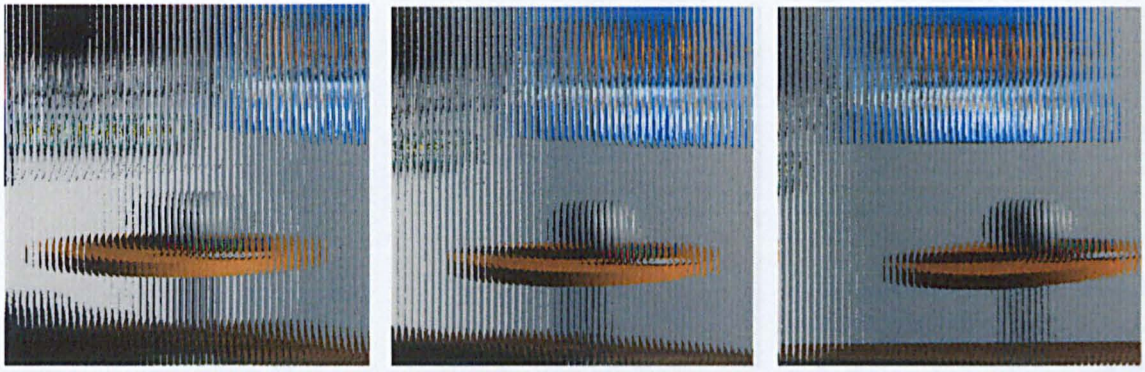


(a) Teapot scene frame 0 (b) Teapot scene frame 9 (c) Teapot scene frame 19

(d) Teapot scene frame 29 (e) Teapot scene frame 39 (f) Teapot scene frame 49

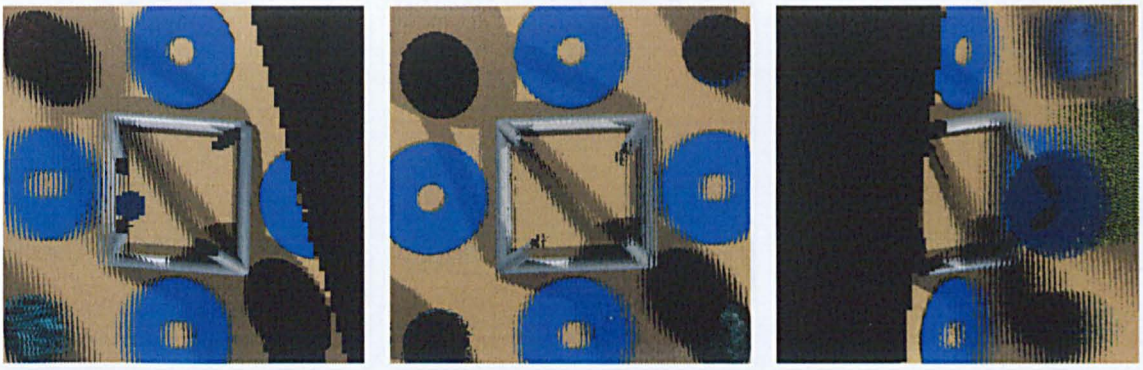
Figure 6.21: Frames of Teapot scene.

(a) Room scene frame 0 (b) Room scene frame 35 (c) Room scene frame 43

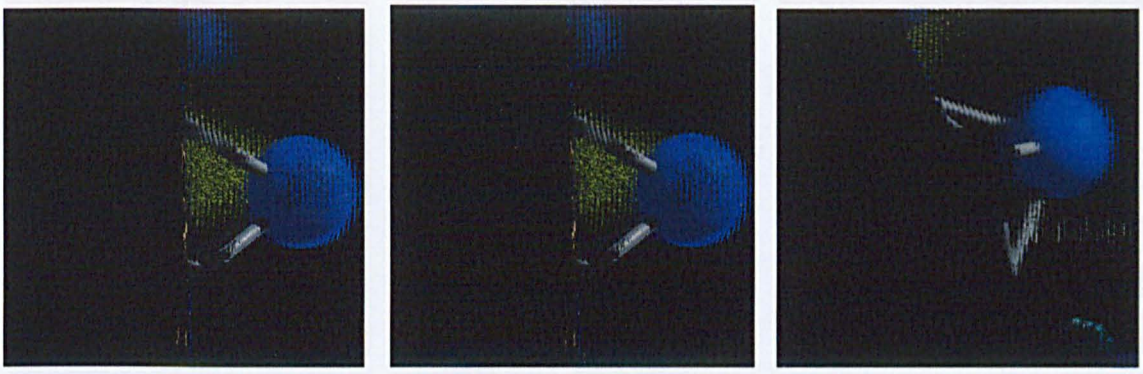


(d) Room scene frame 58 (e) Room scene frame 78 (f) Room scene frame 95

Figure 6.22: Frames of Room scene.

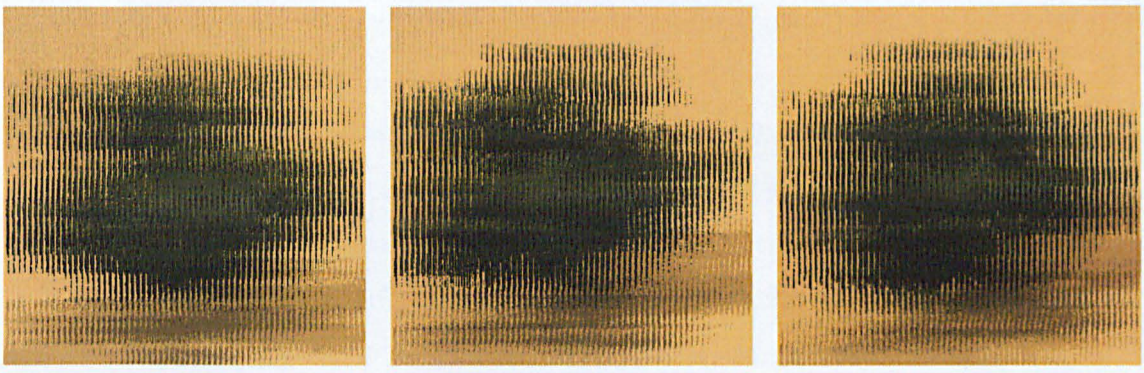


(a) Primitives scene frame 1 (b) Primitives scene frame 19 (c) Primitives scene frame 40

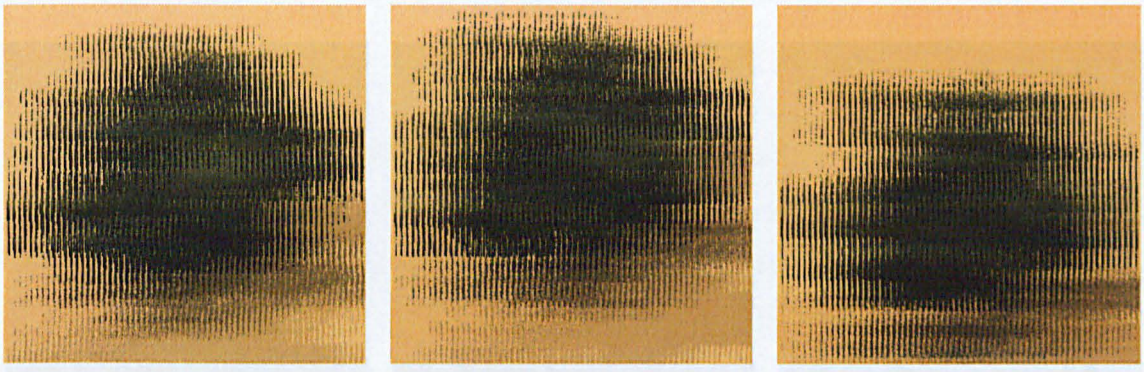


(d) Primitives scene frame 40 (e) Primitives scene frame 60 (f) Primitives scene frame 80

Figure 6.23: Frames of Primitives scene.

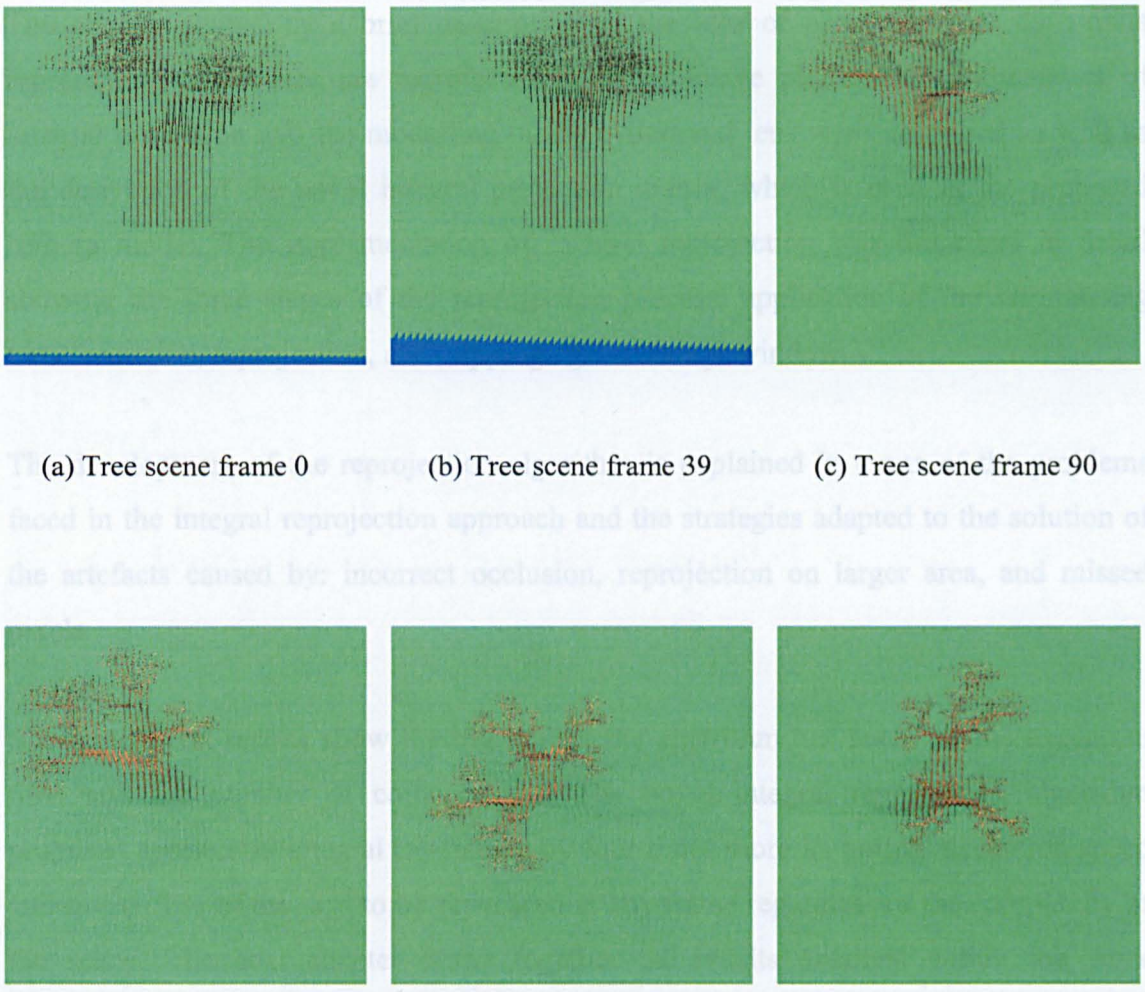


(a) Small-balls scene frame 200 (b) Small-balls scene frame 210 (c) Small-balls scene frame 220



(d) Small-balls scene frame 230 (e) Small-balls scene frame 240 (f) Small-balls scene frame 250

Figure 6.24: Frames of Small-balls scene.



(a) Tree scene frame 0 (b) Tree scene frame 39 (c) Tree scene frame 90

(d) Tree scene frame 106 (e) Tree scene frame 128 (f) Tree scene frame 152

Figure 6.25: Frames of Tree scene.

6.6 Summary

In this chapter a novel approach for speeding up the generation of photo-realistic integral frames has been explained. The approach is named as integral reprojection. The idea of reprojection is to avoid tracing pixels and reuse results from the previous frame for generating the new frame.

The chapter started by a brief description of the idea of projection that the points representing the scenes are reprojected onto the image plane. The mathematics of integral projection and the modelling of the cylindrical lens were described as well as the derivation of the novel integral projection matrix, which is used in the proposed camera model. The implementation of integral reprojection was described in detail showing the three stages of the reprojection process: application of the normalising transformation, reprojection, and clipping against image window.

The development of the reprojection algorithm is explained in terms of the problems faced in the integral reprojection approach and the strategies adapted to the solution of the artefacts caused by: incorrect occlusion, reprojection on larger area, and missed pixels.

Subsequent test results show the big impact the algorithm has made on the execution time and the number of computations. The novel integral reprojection algorithm proposed accelerates integral ray tracing by four times more its normal execution speed and saved 70% of the rays to be ray-traced in any frame regardless of the complexity of the scene. The next chapter draws together the results obtained within the work programme and concludes with recommendations for future work.

Chapter 7

Conclusions and further work

7.1 Conclusions

The present research work has approached the task of accelerating the generation of photo-realistic integral images produced by integral ray tracing.

The 3D integral camera system and the associated image formation and recording process have been presented as well as the computer generation of photo-realistic integral images using integral ray tracing algorithm. Ray tracing algorithm is a computationally exhaustive algorithm, which spawns one ray or more through each pixel of the pixels forming the image, into the space containing the scene. For each ray, intersections with objects of the scene are tested. These tests are termed as intersection tests. This operation is further extended at intersection points lay on the objects of the scene generating shadow, reflection, and transparency rays. Intersection tests are computationally and time exhaustive and overshadow everything else, accounting for the vast bulk of time consumed by ray tracing. In order to produce integral images with acceptable quality, they have to be generated with higher resolution than normal

computer images. Consequently, ray tracing integral images consume more processing time than normal images. The unique characteristics of the 3D-integral camera model has been analysed and it has been shown that different coherency aspects than normal ray tracing can be investigated in order to accelerate the generation of photo-realistic integral images.

The image-space coherence has been analysed describing the relation between rays and projected shadows in the scene rendered. Shadow cache algorithm has been adapted in order to minimise shadow intersection tests in integral ray tracing. Shadow intersection tests make the majority of the intersection tests in ray tracing. Novel pixel-tracing styles are developed uniquely for integral ray tracing to improve the image-space coherence and the performance of the shadow cache algorithm. Experiments show that style 3 and tracing strips of pixels reached up to 41% in time improvement for lenticular sheets and were superior to style 1 (the normal style) by up to 40% in time improvement. Tracing similar pixels and tracing squares of pixels reached up to 18% in time improvement for micro-lens array. However two-dimensional styles did not achieve much improvement over one-dimensional styles. That is because of the diverse sizes of the projected shadows and their positions with respect to the lenticular sheet or the micro-lens array. Also, it has been proven that applying the new styles of pixel-tracing does not affect of the scalability of integral ray tracing running over parallel computers.

The novel integral reprojection algorithm has been developed uniquely through geometrical analysis of the generation of integral image in order to use the tempo-spatial coherence information within the integral frames. A new integral projection matrix has been derived to project intersection points of the scene onto the image plane through the model describing a lenticular lens. The pixels of a new frame were generated using the information leftover from the previous frame. Experiments showed an obvious acceleration of the generation of integral frames. The integral projection algorithm has accelerated the generation of integral frames four times more than its

normal execution speed and has reduced the number of rays to be traced by around 70% regardless of the complexity of the scene.

In a conclusion, a new way of analysing image-space coherence in 3D integral ray tracing through the relationship between spawned rays and projected shadows has been established. Novel pixel-tracing styles, which improve the ray-shadow coherence in integral ray tracing and the performance of shadow cache algorithm has been developed. Acceleration of the photo-realistic integral images generation using the image-space coherence information between shadows and rays in integral ray tracing has been achieved with up to 41% of time saving. A novel way of fast generation of 3D photo-realistic integral frames using reprojection of previously frames has been developed. A new derivation of integral projection matrix for projecting points through an axial model of a lenticular lens has been established. Rapid generation of 3D photo-realistic integral frames has been achieved with a speed four times faster than the normal generation.

7.2 Further work

A number of techniques might further improve the acceleration of 3D-integral image generation using ray tracing. For examples:

Development of a probability scheme for projected shadows. As a pre-processing step this will help to decide which style of pixel-tracing is suitable for a particular region of the integral image. This will allow the division of the image into regions and apply different styles to different regions at the run-time.

Hardware development of 3D-integral graphics accelerators. This might include intersection test acceleration, multi-camera rendering, and third dimension visibility techniques. This will certainly improve the time performance of photo-realistic 3D-integral algorithms.

Tracing missed pixels presented in the thesis could be replaced by different algorithms that will be able to extract the missing information from 3D information of the previous frame without the need to ray-trace all of the missed pixels. This will noticeably accelerate the integral reprojection algorithm.

References

[3dcgi] 3dcgi, http://www.3dcgi.com/coltech/displays/displays.html, April 2nd 2004.

[4D-Vision] X3D Technologies Corp., http://www.4d-vision.de, April 2nd 2004.

[Actualdepth] Deep Video Imaging Ltd, http://www.deepvideo.com, April 2nd 2004.

[Actuality Systems] Actuality Systems, Inc., http://www.actuality-systems.com, April 4nd 2004.

[Aman84] Amanatides, J., "Ray Tracing With Cones," *SIGGRAPH '84*, pp.129-135, January 1984.

[Amateur Holography] http://members.aol.com/gakall/holopg.html, April 2nd 2004.

[Arai98] Arai, J., Okano, F., Hoshino, H. and Yuyama, I., "Gradient Index Lens-Array Method Based On Real-Time Integral Photography For Three-Dimensional Images," *App. Optics*, Vol. 37(11), pp. 2034-2045, 1998.

[Bent80] Benton, S., US Patent US4431265, "*Apparatus For Viewing Stereoscopic Images*," 1980.

[Bent80] Benton, Stephen, A., "Holographic Displays: 1975-1980," *Optical Eng.*, Vol. 19(5), pp. 686-690, 1980.

[Blin76] Blinn, J. F. and M. E. Newell, "Texture And Reflection In Computer Generated Images," *Communication of the ACM*, Vol. (19), pp. 542-547, 1976.

[Bour99] Bourke P, "*Autostereoscopic Lenticular Images,*" http://astronomy.swin.edu.au/~pbourke/stereographics/lenticular/, April 4th 2004.

[Cart00] Cartwright, P., "*Realisation Of Computer Generated Integral Three Dimensional Images,*" De Montfort University, PhD Thesis, 2000.

[Catm74] Catmull, E., "A Subdivision Algorithm For Computer Display Of Curved Surfaces", PhD Thesis, University of Utah,USA, 1974.

[Cohe92] Cohen, M. F. "Is Image Synthesis A Solved Problem?" *Third Eurographics Workshop on Rendering*, pp. 161–167, Bristol, UK, June 1992.

[Davi88] Davies N, "Three Dimensional Imaging Systems: A New Development," *App. Optics*, Vol. 27, pp. 4520-4528, 1988.

[Davi94] Davis, N., McCormick, M., and Brewin, M., "Design And Analysis Of An Image Transfer System Using Microlens Arrays," *Optical Eng.*, Vol. 33(11), pp. 3624-3633, 1994.

[DDD] Dynamic Digital Depth Plc., http://www.ddd.com, April 2nd 2004.

[Dodg97] Dodgson, "Autostereo Displays: 3D Without Glasses," *EID'97*, Surrey.

[DTI] Dimension Technologies Inc, http://www.dti3d.com, April 2nd 2004.

[Dutr96] Dutré, Ph., "*Mathematical Frameworks and Monte Carlo Algorithms for Global Illumination in Computer Graphics*," PhD Thesis, Katholieke Universiteit Leuven, Belgium, 1996.

[Eber94] Ebert, M., Peachey, Perlin and Worley, "*Texturing and Modeling, A Procedural Approach*,"Academic Press, 1994.

[Eric87] Hains, E. A. *et al.*, "A Proposal for Standard Graphics Environments," *IEEE Computer Graphics and Applications,* Vol. 7(11), pp. 3-5, 1987.

[Feib80] Feibush, E. A., M. Levoy, and R. L. Cook, "Synthetic Texturing Using Digital Filters" *SIGGRAPH '80*, Vol 14, pp. 294-301, 1980.

[Fole90] Foley, J. D., A. van DAM, S. K. Feiner, J. F. Hughes, "*Computer Graphics Principles and Practice,*" Second Edition, Addison-Wesley, 1990.

[Gabo48] Gabor, D., "A New Microscope Principle," *Nature*, No. 161, pp. 777-779, 1948.

[Gabo49] Gabor, D., "Microscopy By Reconstructed Wavefronts", *Pro. Roy. Soc.*, A194, pp. 454-487, 1949.

[Genex] Genex Limited, http://www.genextech.com/, April 2nd 2004.

[Glas98] Andrew Glassner, editor. "*An Introduction to Ray Tracing*," Academic Press, 1989.

[Glas90] Glassner, A., editor, "*Graphics Gems*," Academic Press, 1990.

[Glas91] Glassner, A., editor, "*Graphics Gems II*," Academic Press, 1991.

[Glas92] Glassner, A., editor, "*Graphics Gems III*," Academic Press, 1992.

[Gora84] Goral, C. M., K. E. Torrance, D. P. Greenberg, and B. Battaile. "Modeling The Interaction Of Light Between Diffuse Surfaces," *Proceedings of SIGGRAPH '84 In Computer Graphics*, Vol. 18, pp. 213–222, July 1984.

[Gree86] Greene, N, "Environment Mapping And Other Applications Of World Projections," *IEEE CG & A*, Vol. 6(11), pp. 21-29, 1986.

[Hain86] Haines, E. A., "*The Light Buffer: A Shadow Testing Accelerator*," Master Thesis, Cornell University, January 1986.

[Hain86b] Hains, E. A. *et al*, "The Light Buffer: A Shadow-Testing Accelerator," *IEEE CG & A,* Vol. 6, pp. 6-16, 1986.

[Hall83] Hall, R. A. and D. P. Greenberg, "A Testbed for Realistic Image Synthesis," IEEE CG and A, Vol. 3(8), pp. 10-20, November 1983.

[Hall97] Halle, M. W. *et al*, "Fast Computer Graphics Rendering For Full Parallax Spatial Displays," *Practical Holography XI and Holographic Materials III Proc. Of the SPIE*, Vol. 3011, 1997.

[Hari02] Hariharan, P., "*Basics of Holography*," Cambridge University, 2002.

[Harm96] Harman, P., " Autostereoscopic Display System", *Proc. SPIE*, Vol. 2653, pp. 56-64, 1996.

[Heck84] Heckbert, P. S. and P. Hanrahan. "Beam Tracing Polygonal Objects," *SIGGRAPH '84*, Vol. 18, pp. 119-127, July 1984.

[Heck86] Heckbert, P. S., "*Fundamentals of Texture Mapping and Image Warping*," Master Thesis, University of California-Berkeley, USA, June 1986.

[Heck 91] Heckbert, P. S. and H. P. Moreton "Interpolation For Polygon Texture Mapping And Shading," *State of the Art in Computer Graphics: Visualization and Modeling*, pp.101-111. Springer-Verlag, 1991.

[Heck94] Heckbert, P., editor, "*Graphics Gems IV*," Academic Press, 1994.

[HyperGraph] HyperGraph, A Project of the ACM SIGGRAPH Education Committee, http://www.siggraph.org/education/materials/HyperGraph/hypergraph.htm, April1st 2004.

[Ives31] Ives, H. E., "Optical Properties of a Lippmann Lenticulated Sheet," *J. Opt. Soc. Am.*, Vol. 20, pp.171-176,1931.

[Kaji86] Kajiya, J.T., "The Rendering Equation," *SIGGRAPH '86*, Vol 20, pp. 143–150, August 1986.

[Kasp87] Kasper, J. and S. Feller, "*The Complete Hologram Book*," Prentice-Hall, 1987.

[Kay86] Kay, T. and J. Kajiya, "Ray Tracing Complex Scenes" *SIGGRAPH '86*, pp. 268-278, August 1986.

[Kirk87] Kirk D. B., "The Simulation Of Natural Features Using Cone Tracing," *The Visual Computer*, Vol. 3(2), pp. 63-71, 1987.

[Kita01] Kitamura, Y., T. Konishi, S. Yamamoto and F. Kishino, "Interactive Stereoscopic Display for Three or More Users," *Proceedings of The 28th Annual Conference on Computer Graphics and Interactive Techniques*, pp. 231 - 240, August 2001.

[Klei96] Kleiman, S. and Smaalders, "*Programming with Threads*," SunSoft Press and Prentice Hall, 1996.

[Kok94] Kok, A. J. F., "*Ray Tracing and Radiosity Algorithms for Photorealistic Image Synthesis*," PhD Thesis, Delft University of Technology, The Netherlands, 1994.

[Krat72] Kratomi, S., US Patent US3737567, "*Stereoscopic Apparatus Having Liquid Crystal Filter Viewer*," 1972.

[Leit63] Leith, E. and J. Upatnieks, "Wavefront Reconstruction With Continuour-Tone Objects," *J. Opt. Soc. Am.*, Vol. 53(12), pp. 1377-1381, 1963.

[Lewi71] Lewis, D. Jordan, Verber, M. Carl and A. McGhee, "True Three-Dimensional Display," *IEEE Trans. on Electronic Devices*, Vol. 18, pp. 724-732. 1971.

[Lipp08] Lippmann, G.,"La Photographie Integrale," Comtes Rendus, *Academie des Sciences*, Vol.146, pp.446-451,1908.

[Mano99] Manolache, S., A. Aggoun, M. McCormick and N. Davies , "A Mathematical Model Of A 3d-Lenticular Integral Recording System," *Proceedings of IEEE Vision, Modeling and Visualization Conference,* Erlangen, pp. 51-58, 1999.

[McAl93] McAllister, D., "Stereo Computer Graphics And Other True 3D Technologies," Princeton University, 1993.

[McCo92] McCormick, M., N. Davies and E. Chovanietz, "Restricted Parallax Images 3D TV", *IEE Colloq. 'Stereoscopic Television'*, No. 173, 3/1-3/4, November 1992.

[McCo94] McCormicK M., "Examination Of The Requirements For Autostereoscopic, Full Parallax, 3D TV," *IEE Conf. Pub.* No. 397, pp. 477-482, 1994.

[McCo95] McCormick M. and N. Davies, "Full Natural Colour 3D Optical Models By Integral Imaging," *Proceedings of IEE 4th International Conference on Holographic Systems, Components and Applications,* No. 379, pp. 237-242, Switzerland, September 1995.

[Min01] S. Min *et al*, "Three-Dimensional Display System Based On Computer-Generated Integral Photography," *Stereoscopic Display and Virtual Reality Systems VIII Proc. Of the SPIE*, Vol. 4297, pp. 187-195, 2001.

[Moto95] Motoki T., H. Isono and I. Yuyama , "Present Status of Three-Dimensional Television Research," *Proc. IEEE'83*, pp. 1009-1021, 1995.

[MPI] Max-Planck-Institut für Informatik, http://www.mpi-sb.mpg.de/index.html, April1st 2004.

[Naeu01] Naemura T., T. Yoshida and H. Harashima, "3-D Computer Graphics Based on Integral Photography," *Optics express*, Vol. 8(2), pp. 255-262, 2001.

[Nich96] Nichols, B., and Farrell, "*Pthreads Programming*," O'Reilly and Associates, 1996.

[Okan98] Okano F., H. Hoshino, J. Arai and I.Yuyama, "Real-Time Pickup Method for a Three-Dimensional Image Based on Integral Photography," *Applied Optics*, Vol. 6(7), pp.1598-1603, 1998.

[Okos76] Okoshi T, "*Three Dimensional Imaging Techniques,*" Academic Press, London, 1976.

[Osam02] Youssef, O. H. *et al*, "Pixels Grouping And Shadow Cache For Faster Integral 3d Ray Tracing,"*Stereoscopic Displays and Virtual Reality Systems IX Proc. Of the SPIE,* Vol. 4660, pp. 123-134, May 2002.

[Outw99] Outwater, C. and V. Hamersveld , "*Practical Holography*," Dimensional Arts Inc. 1995-99, http://www.holo.com/holo/book/book1.html. April 2nd 2004.

[Paet95] Paeth, A., editor, "*Graphics Gems V*," Academic Press, 1995.

[Philips] Philips, http://www.research.philips.com, April 5th 2004.

[Phon73] Phong, B. T., "*Illumination for Computer Generated Images,*" PhD Thesis, University of Utah, 1973.

[Phon75] Phong, B. T., "Illumination For Computer Generated Images," *Communication of the ACM*, Vol. 18, pp. 311-317, 1975.

[Romn69] Romney, G.W., "*Computer Assisted Assembly and Rendering of Solids*," PhD Thesis, University of Utah, USA, 1969.

[Rubi80] Rubin, S. and T. Whitted, "A Three-Dimensional Representation For Fast Rendering Of Complex Scenes," *SIGGRAPH '80*, Vol. 14(3), pp.110-116, July 1980.

[Shin87] Shinya, M., T. Takahashi, S. Naito, "Principles And Applications Of Pencil Tracing," *SIGGRAPH '87*, Vol. 21(4), pp.45-54, July 1987.

[Shir91] Shirley, P., "*Physically Based Lighting Calculations for Computer Graphics*," PhD Thesis, University of Illinois, USA, 1991.

[Shir94] Shirley, P. and G. Sakas. "Results Of The 1994 Survey On Image Synthesis," *Fifth Eurographics Workshop on Rendering*, pp. 3–6, Darmstadt, Germany, June 1994.

[Sieg95] Siegel, M., V. Grinberg, A. Jordan, J. McVeigh, G. Podnar, S. Safier and S. Sriram, " Software for 3D-TV and 3D-Stereoscopic Computer Workstations," *Proc. of the International Workshop on Stereoscopic and Three Dimensional Imaging*, pp. 251-260, 1995.

[Stereographics] Stereographics, http://www.stereographics.com, April 5th 2004.

[Stone98] Stone, J. E., "*An Efficient Library For Parallel Ray Tracing And Animation,*" Master Thesis, University Of Missouri-Rolla, USA, 1998.

[Taub02] Taub, A. "Will 3-D Ever Catch On?" *The New York Times*, July 18th 2002.

[Tets94] Tetsutani, N., K. Omura, F. Kishino, "Wide-Screen Autostereoscopic Display System Employing Head-Position Tracking," *Opt. Eng.*, Vol. 33(11), pp. 3690-3697, November 1994.

[Valy66] Valyus, N., "*Stereoscopy,*" Focal Press, London 1966.

[Wan04] Wan, M., N. Zhang, H. Qu, and A. Kaufman, "*Interactive Stereoscopic Rendering of Volumetric Environments,*" *IEEE Trans. on V & CG*, Vol. 10(1), pp.15 - 18, 2004.

[Watk93] Watkins, C., A. Sadun, and S. Marenka, "*Modern Image Processing*," Academic Press, 1993.

[Watt92] Watt, A. and M. Watt, "*Advanced Animation and Rendering Techniques*," ACM Press, 1992.

[Watt93] Watt, A., "*3D Computer Graphics*," Addison Wesley, 1993.

[Wegh84] Weghorst, H., G. Hooper and D. Greenberg, "Improved Computational Methods For Ray Tracing," *ACM Transactions On Graphics*, Vol. 3(1), pp.52-69, January 1984.

[Whit80] Whitted, T. "An Improved Illumination Model For Shaded Display," *Communication of the ACM*, June 1980.

[Wu03] Wu, C., "*Depth measurement in integral images,*" PhD Thesis, De Montfort University, 2003.

[Yano02] Yano, S., I. S. Mitsuhashi and H. Thwaite, "A Study Of Visual Fatigue And Visual Comfort For 3D HDTV/HDTV Images," *Displays*, Vol.23(4), pp.191-201, 2002.

Appendix
A Mathematical Framework for Global Illumination

This appendix introduces the basic terms and definitions needed to formulate the global illumination problem in a concise mathematical form. Since photo-realistic image rendering solves the global illumination problem in three-dimensional environments, it is necessary to describe the geometry of that environment. Therefore the appendix starts by formally defining some notations used to describe several concepts in geometry.

Light is electromagnetic radiation at wavelengths visible to the human eye [Dutr96]. It is therefore necessary to define terms and concepts of radiometry. The relevant units, which are needed for the global illumination problem, will be defined.

A.1 Geometry

A.1.1 Surface points

The three-dimensional scene to be rendered is usually composed of solid objects. These objects are all bounded by surfaces, which can be described by their equations in three-dimensional space. The surfaces delineate the boundary between the solid objects and the surrounding space, through which light can pass freely. Surface points are points in

the three-dimensional space that belong to one of the surfaces describing an object. The entire set of all surface points is denoted by A. In each surface point x, a surface normal n_x can be constructed. Some problems can arise at discontinuities in the surface (e.g. the edges of a cube), where a normal cannot be formally defined. The differential surface area around a point x is written as dA_x as shown in figure A.1. In the absence of a participating medium, which can absorb or scatter energy, energy transport between surface points only is considered in global illumination problems. If the light transport between any two arbitrary differential surface areas present in the scene, can be computed, a complete knowledge of the energy distribution can be gained, and thus an image can be rendered from any viewpoint.

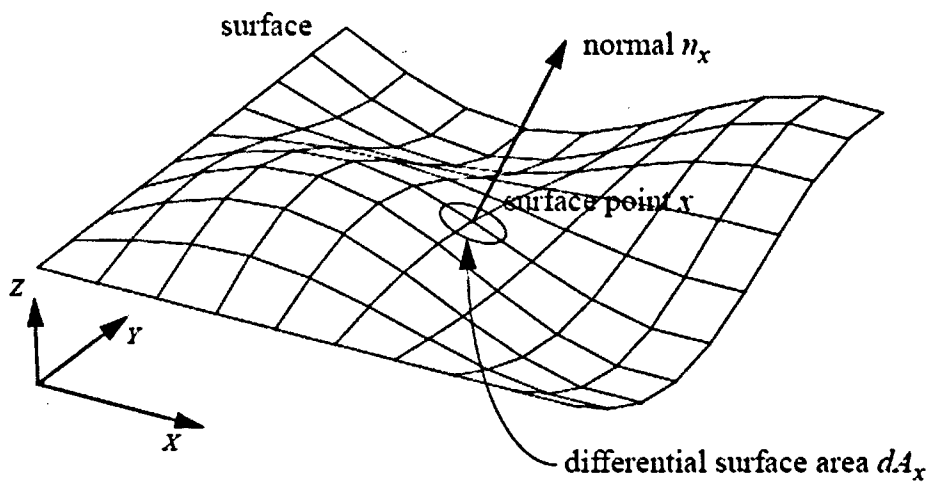


Figure A.1: Surfaces in a three-dimensional scene [Dutr96].

A.1.2 Directions & solid angles

Directions in a three-dimensional vectorspace can be represented in the same way as points or vectors. However, in the context of global illumination, directions are often thought of as being defined on a sphere surrounding a surface point. All directions

surrounding a surface point can be positioned on a sphere around that point. A hemisphere around a surface point is the half of the surrounding sphere, consisting of all directions pointing to the same side of the surface. In other words, a hemisphere encompasses all directions for which the cosine of the angle with the normal has the same positive sign. A hemisphere around point x and a differential solid angle around direction Θ are denoted as Ω_x and $d\omega_\Theta$ respectively. Each direction can be described by two angles θ and φ using a spherical coordinate system. These spherical coordinates are usually relative to the normal vector n_x for surface point x, and to an arbitrary chosen axis a from which φ is measured. The solid angle $d\omega_\Theta$ can then be expressed as $\sin\theta d\theta d\varphi$ as shown in figure A.2.

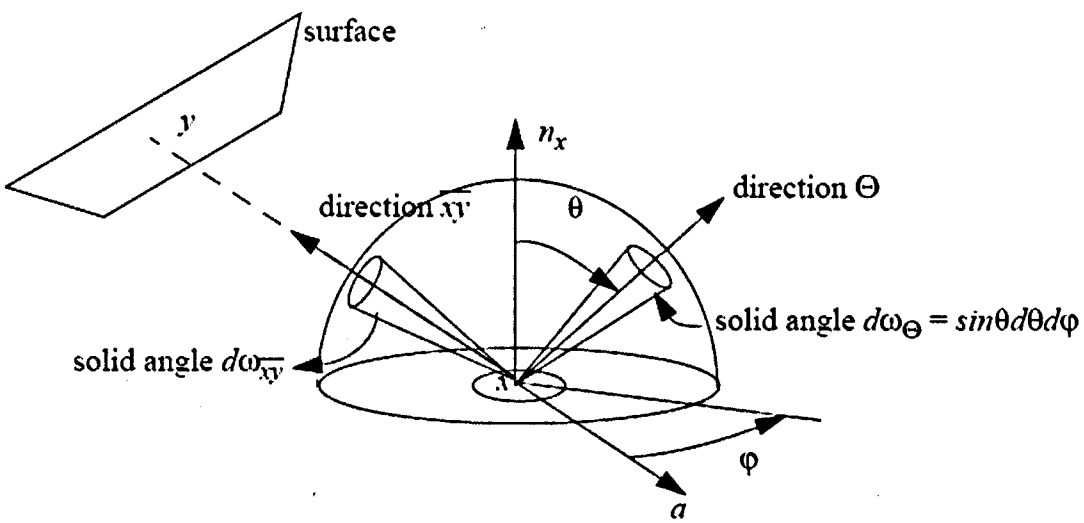


Figure A.2: Spherical coordinates [Dutr96].

The differential solid angle $d\omega_\Theta$ shown in figure A.3, can also be expressed in terms of a differential area visible along the direction of the differential solid angle:

$$d\omega_{\Theta} = d\omega_{\overline{xy}} = \frac{\cos\theta_y dA_y}{r_{xy}^2}$$

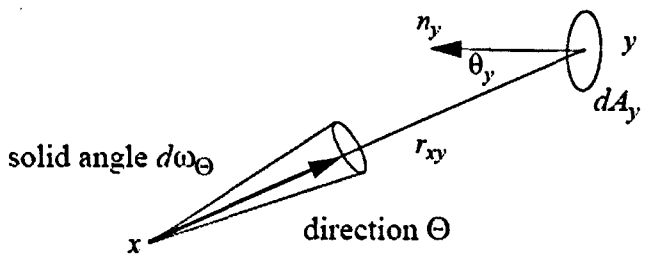


Figure A.3: Transformation of a differential solid angle to a differential surface area [Dutr96].

Since Θ is a vector, the opposite direction of Θ is written as $-\Theta$.

2.1.3 Visibility and the ray-casting function

Visibility determination is an important concept in global illumination algorithms, because it determines what surfaces can possibly exchange energy with each other. Given two arbitrary points x and y in three-dimensional space, the visibility function $V(x, y)$ equals 1 or 0, depending on whether those two points are mutually visible to each other. Mutual visibility means that there are no occluding objects or surface points. Even if some of the occluding objects are transparent, they are considered to be blocking visibility, because they cause a change in energy transfer at their surface.

Given a point x and a direction Θ, one often wants to find the closest surface point visible from x along direction Θ (See figure 2.4), for a given geometry of the three-dimensional scene. It is obvious that there can only be one such point. This closest

visible point can therefore be considered as a function of x and Θ. This function is called the ray-casting function and is denoted as $r(x, \Theta)$ and is formally defined by equation (A.1). The name ray-casting refers to the fact that algorithms usually determine the nearest visible point by casting a ray from point x in direction Θ, searching for the nearest intersection point of this ray with other surfaces in the scene.

$$\begin{aligned} r(x,\Theta) &= x + t_{inf} \cdot \Theta \\ t_{inf} &= inf\{t > 0 : (x + t \cdot \Theta) \in A\} \end{aligned} \tag{A.1}$$

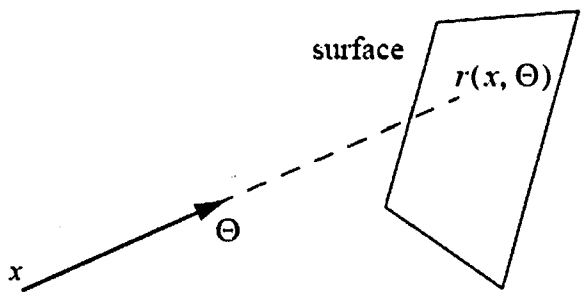


Figure A.4 Ray-casting function [Dutr96].

If there is no point belonging to A that can be designated as the closest visible point (which can possibly occur if the three-dimensional scene is not closed) the ray-casting function is undefined.

If x itself is a surface point, then x in its turn is the closest visible surface point to $r(x, \Theta)$ in the opposite direction of Θ as shown in figure A.5:

$$\begin{aligned} &\text{if } \; y = r(x,\Theta) \;\; \text{and} \;\; x \in A \\ \Rightarrow\; &x = r(y,-\Theta) \;\; \text{or} \;\; x = r(r(x,\Theta),-\Theta) \end{aligned}$$

The ray-casting function is only dependent on the geometry of the scene, and its evaluation is a classic problem in rendering algorithms and computer graphics in general. In most algorithms, the evaluation of the ray-casting function often requires a lot of thought and optimization, as a clever implementation or optimization scheme can drastically improve execution times. Ray tracing algorithms use the ray-casting function in order to find the closest intersection point for any ray to be traced; radiosity algorithms often use this function in order to numerically compute form factors between surface patches.

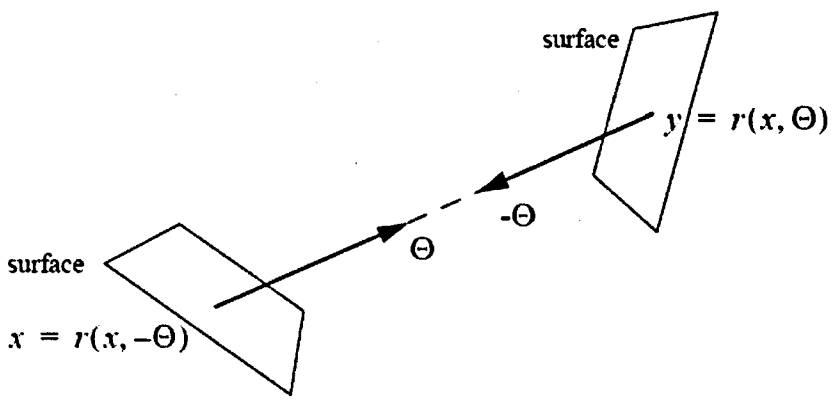


Figure A.5 Reciprocity of the ray-casting function [Dutr96].

A.2 Radiometry

This section deals with the different quantities and units regarding illumination.

A.2.1 Radiant energy

The basic unit of light energy used in global illumination algorithms is radiant energy. It is denoted Q and is measured in Joule.

A.2.2 Radiant flux

The radiant energy flowing through (or leaving or arriving at) a surface per unit time is called radiant flux Φ, and is expressed in Watt.

$$\Phi = \frac{dQ}{dt}$$

Radiant flux is the quantity most used by global illumination algorithms, since the interest is in the time equilibrium of the transport of light energy. Time-dependent energy distributions are usually not considered.

A.2.3 Exiting radiance and irradiance

Another useful measure is the incident or departing flux per unit of surface area. These measures are called irradiance (E) and exiting radiance (M) respectively. Exiting radiance is also often called radiosity (B) in the context of global illumination algorithms.

$$E = \frac{d\Phi}{dA} \qquad M = B = \frac{d\Phi}{dA}$$

A.2.4 Radiant intensity

Flux can also be measured per unit solid angle. This is suitable for the case of describing flux arriving at or leaving point sources. The term radiant intensity (I) is used to measure this quantity.

$$I = \frac{d\Phi}{d\omega}$$

A.2.5 Radiance

Finally, the measures of flux per unit area and per unit solid angle can be combined in one single term. The power arriving at or leaving from a surface dA in a certain direction, per unit solid angle $d\omega$ around that direction, per unit projected area perpendicular to the direction of travel, is called radiance L:

$$L = \frac{dE}{d\omega} = \frac{d^2\Phi}{d\omega dA^{\perp}} = \frac{d^2\Phi}{d\omega dA\cos\theta}$$

where θ is the angle between the surface normal and the direction of travel under consideration.

Radiance can intuitively be described as perceived intensity. Observers, or detectors such as cameras, indeed perceive intensity as irrelevant of the size of the surfaces, or the aperture of a given solid angle. If two surfaces with areas of 10 m^2 and 1 m^2 are considered, both emitting 1000 Watt, then the second surface looks more bright, due to a higher amount of energy per unit surface area. The same consideration can be made for power distributed over small or large solid angles.

Since radiance at a point x is defined with respect to a certain direction, careful usage of notations must be applied, since radiance can be considered as well as an incident or as an exiting measure at x, regardless of the direction specified as shown in figure A.6. The following notations is adopted by Dutré [Dutr96]:

- $L(x \rightarrow \Theta)$: radiance leaving point x in direction Θ.
- $L(x \leftarrow \Theta)$: radiance arriving at point x from direction Θ.
- $L(x \rightarrow y)$: radiance leaving point x towards point y.
- $L(x \leftarrow y)$: radiance arriving at point x, coming from the direction of point y.

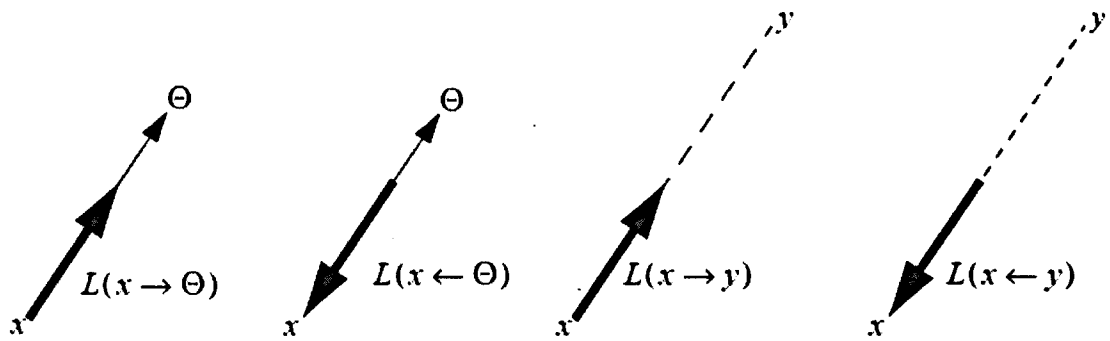


Figure A.6: Different notations for exiting and incident radiance [Dutr96].

An important property of radiance is its invariance along straight paths. The radiance leaving point x directed towards point y is equal to the radiance arriving at point y from the direction in which point x is observed. In other words: $L(x \rightarrow y) = L(y \leftarrow x)$. This can be proven by considering two differential surface areas, and by computing energy transport between them as shown in figure A.7.

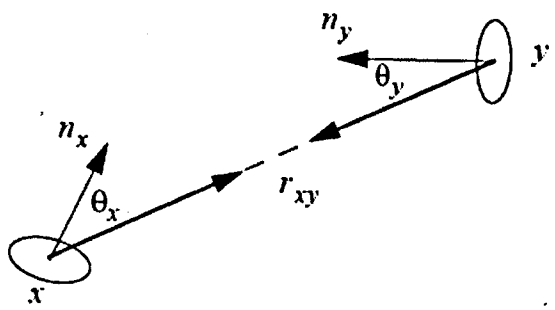


Figure A.7: Energy transport between two differential surfaces [Dutr96].

Due to the definition of radiance, the total flux or power, which is leaving differential surface area dA_x arriving at dA_y, can be written as:

$$d^2\Phi = L(x \rightarrow y)\cos\theta_x d\omega_{\overline{xy}} dA_x$$

where $\overline{xy}$ is the direction pointing from x to y, $d\omega_{\overline{xy}}$ is the solid angle under which dA_y is seen from x. The power that arrives at area dA_y from area dA_x can be expressed in a similar way:

$$d^2\Phi = L(y \leftarrow x)\cos\theta_y d\omega_{\overline{yx}} dA_y$$

Assuming that no energy loss occurs between the two differential surfaces, and that there are no external light sources adding to the power arriving at dA_y, then due to conservation of energy, all energy that leaves the surface dA_x must arrive at the surface dA_y:

$$L(x \rightarrow y)\cos\theta_x d\omega_{\overline{xy}} dA_x = L(y \leftarrow x)\cos\theta_y d\omega_{\overline{yx}} dA_y$$

$$L(x \rightarrow y)\cos\theta_x \frac{\cos\theta_y dA_y}{r_{xy}^2} dA_x = L(y \leftarrow x)\cos\theta_y \frac{\cos\theta_x dA_x}{r_{xy}^2} dA_y$$

$$L(x \rightarrow y) = L(y \leftarrow x)$$

As a consequence, radiance does not attenuate with distance, and is invariable along straight paths of travel. Since most light receivers, such as cameras or the human eye, are sensitive to radiance, this property explains why the perceived colour or brightness of an object does not change with distance. If we allow a participating medium to be present between the surfaces, which can absorb and scatter energy, the above properties of radiance are no longer valid.

From the above observations, it follows that incident or exiting radiance is known at all surface points, the radiance distribution for all points in a three-dimensional scene is known. Most of the algorithms used in global illumination limit themselves to compute the radiance values at surface points (still assuming the absence of any participating medium).

A.2.6 Bi-directional Reflectance Distribution Function

Materials interact with light in different ways, and different materials have different appearances given the same lighting conditions. Some materials appear as mirrors, others appear as diffuse surfaces. The reflectance properties of a surface are described by a reflectance function, which models the behaviour of interaction of light reflecting at a surface.

The bi-directional reflectance distribution function (BRDF) is the most general expression of reflectance of a material. The BRDF is defined as the ratio between differential radiance reflected in an exiting direction, and incident irradiance through a differential solid angle; or more precisely, the BRDF is defined as the derivative of reflected radiance to incident irradiance. See figure A.8.

$$f_r(x, \Theta_i \rightarrow \Theta_r) = \frac{dL(x \rightarrow \Theta_r)}{dE(x \leftarrow \Theta_i)} = \frac{dL(x \rightarrow \Theta_r)}{L(x \leftarrow \Theta_i)\cos\theta_i d\omega_{\Theta_i}} \tag{A.2}$$

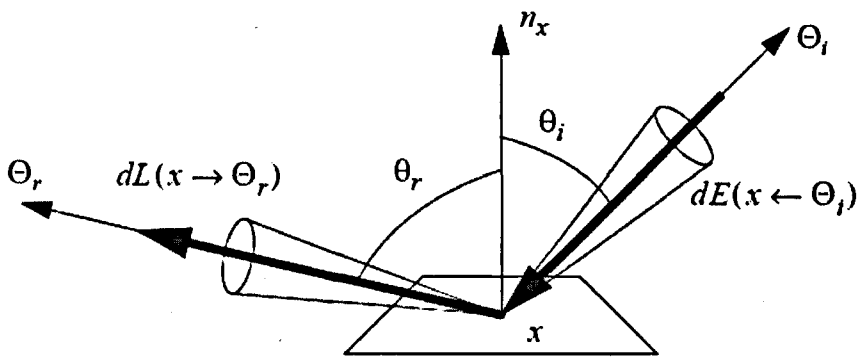


Figure A.8: Geometry for the BRDF [Dutr96].

The BRDF can take any positive value, and varies with wavelength. The BRDF has some interesting properties:

1. The value of the BRDF will remain unchanged if the incident and exiting directions are interchanged. This property is also called the Helmholtz reciprocity; a principle, which says that paths followed by light, can be reversed [Dutr96].

$$f_r(x, \Theta_i \rightarrow \Theta_r) = f_r(x, \Theta_r \rightarrow \Theta_i) \tag{A.3}$$

   Because of the reciprocity property, Dutré uses a double arrow to indicate the fact that the two directions may be freely interchanged: $f_r(x, \Theta_r \leftrightarrow \Theta_i)$.

2. Generally, the BRDF is anisotropic. That is, if the surface is rotated about the surface normal, the value of f_r will change.

3. The value of the BRDF for a specific incident direction is not dependent on the possible presence of irradiance along other incident angles. Therefore, the BRDF as defined above behaves as a linear function with respect to all incident directions. In order to know the total reflected radiance due to some irradiance distribution over the hemisphere around an opaque, non-emissive surface point, equation A.2 has to be integrated over the surrounding hemisphere, and this provides the equation, which is referred to as the reflectance equation:

$$\begin{aligned} dL(x \rightarrow \Theta_r) &= f_r(x, \Theta_i \rightarrow \Theta_r) dE(x \leftarrow \Theta_i) \\ L(x \rightarrow \Theta_r) &= \int_{\Omega_x} f_r(x, \Theta \leftrightarrow \Theta_r) dE(x \leftarrow \Theta) \\ L(x \rightarrow \Theta_r) &= \int_{\Omega_x} d\omega_\Theta f_r(x, \Theta \leftrightarrow \Theta_r) L(x \leftarrow \Theta) \cos(n_x, \Theta) \end{aligned} \tag{A.4}$$

   where $\cos(n_x, \Theta)$ is the cosine of the angle formed by the vectors n_x and Θ.

Depending on the nature of the BRDF, the material will appear as a diffuse surface, as a mirror, as a glossy surface as shown in figure A.9, or can exhibit any behaviour described by the BRDF. The more common encountered types of BRDF, as used in photo-realistic rendering, are listed below:

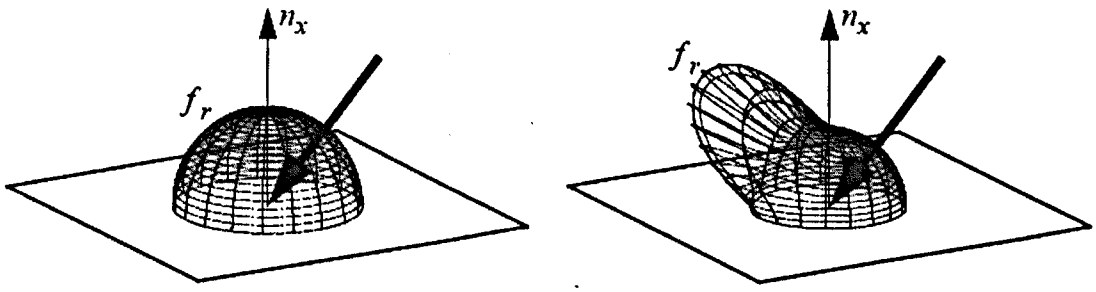


Figure A.9: Diffuse and glossy surfaces [Dutr96].

A.2.6.1 Diffuse surfaces

Some materials reflect light in a uniform way over the entire reflecting hemisphere. That is, given an irradiance distribution, the reflected radiance is independent of the exiting direction. Such materials are called diffuse reflectors, and the value of their BRDF is constant for all values of Θ_r. To an observer, a diffuse material looks the same from all possible directions.

A.2.6.2 Specular surfaces

Other materials can be considered as perfect specular surfaces and only reflect light in one specific direction according to Snell's law, stating that the incident and exiting direction make equal angles to the surface's normal. Perfect specular surfaces are merely an ideal mathematical concept. A perfect specular surface has only one exiting direction for which the BRDF is different from 0, which implies that the value of the BRDF along that direction is infinite. Real materials can exhibit this behaviour very closely, but are nevertheless not "ideal reflectors" as defined above.

A.2.6.3 Glossy surfaces

Most surfaces, however, are neither ideally diffuse nor ideally specular, but act as a combination of both reflectance behaviours. Their BRDF is often difficult to model with

analytical formulas. In the literature, these surfaces are often called glossy surfaces, but the terms diffuse-like, specular-like, etc. are also used.

A.2.6.4 Transparent surfaces

Strictly speaking, the BRDF is defined over the entire sphere of directions (4π steradians) around a surface point. This is important for transparent surfaces, since these surfaces can "reflect" light over the entire sphere. The transparent side of the BRDF can also behave as diffuse, specular or glossy, depending on the transparency characteristics of the material.

List of Publications

1. Youssef, O. H. *et al*, "Pixels Grouping And Shadow Cache For Faster Integral 3D Ray Tracing," *Stereoscopic Displays and Virtual Reality Systems IX Proc. Of the SPIE,* Vol. 4660, pp. 123-134, May 2002.

2. Youssef, O. H. *et al*, "Coherent Grouping of Pixels for Faster Shadow Cache in Integral 3D Ray-Tracing," Submitted to the *Journal of Electronic Imaging*, October 2003.

3. Youssef, O. H. *et al*, "Interactive Autostereoscopic Rendering using Reprojection," To be submitted to *IEE Proc. Of Vision, Image & Signal Processing*.